НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

УДК: 044

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“___” _____2019 р.

Дипломний проект
на здобуття ступеня бакалавра

з напряму підготовки *6.050101 «Комп’ютерні науки»*

на тему: «*Автоматизована рекомендаційна система для вибору театральних подій*»

Виконав: студент *4* курсу, групи *ІС-51*
Штик Василь Леонідович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доц., к.т.н., доц. Тєлишева Т.О.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації *ст.викл. Москаленко Н.В.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент Штик В.Л. _____
(підпис)

Київ – 2019 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з п'яти розділів, містить 65 сторінок, 36 рисунків, 31 таблицю, 1 додаток та 12 джерел.

Дипломний проект присвячений розробці рекомендованої системи з підбору театральних вистав з метою забезпечення інформаційних процесів по інформуванню та взаємодії користувачів.

У інформаційному забезпеченні було визначено вхідні та вихідні дані до комплексу задач, була розроблена структура бази даних для збереження даних, яка відповідає поставленим цілям проекту.

У програмному забезпеченні було описано основні засоби розробки комплексу задач, висунуті вимоги до технічного забезпечення, обрано та обґрунтовано архітектуру програмного забезпечення.

У технологічному забезпеченні було надано інструкцію користувача та проведене тестування комплексу задач.

Практичним значенням даного дипломного проекту є популяризація відвідування театру шляхом забезпечення зручного процесу бронювання місць на театральні події з використанням рекомендаційної системи.

ТЕАТР, CRM-СИСТЕМА, РЕКОМЕНДАЦІЙНА СИСТЕМА, ВІДВІДУВАЧ, БРОНЮВАННЯ, СМС-ПОВІДОМЛЕННЯ

ABSTRACT

The structure and scope of work. Explanatory notebook consists of five sections, containing 65 pages, 36 figures, 31 tables, 1 supplement and 12 sources.

The diploma project is devoted to the development of a recommended system for selecting theatrical performances in order to provide information processes for informing and interacting users.

In the information provision, the input and output data were identified in a set of tasks, a database structure for data storage was developed that corresponds to the objectives of the project.

The software described the main tools for developing a set of tasks, the requirements for technical support, the architecture of the software was selected and substantiated.

In the technology provided, a user manual was described and a set of tasks was tested.

The practical significance of this diploma project is the popularization of theater visits by providing a convenient process of booking places for theatrical events using the advisory system.

THEATER, CRM SYSTEM, RECOMMENDATION SYSTEM, VISITOR, RESERVATION, SMS-MESSAGE

ЗМІСТ

ВСТУП

Перший театр було засновано у Греції 534 р. до н. е., коли афінський поет Феспід під час хорового виступу використав одного актора. Пізніше кількість акторів виросла до трьох, а ще пізніше на сцені першого театру грало близько десяти акторів. Це справило велике враження на глядачів та театр почав поширюватись за рамки Греції.

Античний театр налічував лише два жанри: комедію та трагедію, що писалися з міфів. Перші театральні вистави не мали декорацій чи спеціальних костюмів, проте глядачі з захопленням дивились вистави. Театр став для людей місцем, де з певною сатирою можливо виразити своє незадоволення на ту чи іншу ситуацію. Саме захоплення в народі зробило театр популярним в античні часи. За часи Ренесансу ходити в театр вважалось престижно, тому всі високопоставлені персони не пропускали жодного виступу. У наші часи театр став для глядачів більше як хобі. Люди з великим задоволенням ходять на вистави своїх улюблених драматургів, з великим натхненням обговорюють виставу та акторську гру.

Тобто, у всі часи глядач займав, ледве, не основне місце в житті театру. Всі старання театральної трупи були направленні на задоволення бажань глядача. Саме глядач є споживачем контенту, що надається театром.

Для покращення сервісу для глядача, виникло рішення розробити рекомендаційну системи з підбору театральних подій.

Метою даного дипломного проекту є розробка рекомендаційної системи, що полегшить глядачу вибір театральних вистав.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Театр (грец. Θέατρον - основне значення - місце для видовищ, потім - видовище, від θεάομαι - дивлюся, бачу) - видовищний вид мистецтва, що представляє собою синтез різних мистецтв - літератури, музики, хореографії, вокалу, образотворчого мистецтва та інших і володіє власною специфікою: відображення дійсності, конфліктів, характерів, а також їх трактування та оцінка, твердження тих чи інших ідей тут відбувається за допомогою драматичної дії, головним носієм якого є актор[1].

Родове поняття «театр» включає в себе різні його види: драматичний театр, оперний, балетний, лялькoвий, театр пантоміми та ін.

У всі часи театр представляв собою мистецтво колективне; в сучасному театрі в створенні вистави, крім акторів і режисера (диригента, балетмейстера), беруть участь художник-сценограф, композитор, хореограф, а також бутафори, костюмери, гримери, робітники сцени, освітлювачі.

Походження терміна пов'язане з давньогрецькою античним театром, де саме так називалися місця в залі для глядачів (від грецького дієслова «теаомай» - дивлюся). Тобто від початку глядач займав, ледве, не основне місце в житті театру. Всі старання театральної трупи були направленні на задоволення бажань глядача. Саме глядач є споживачем контенту, що надається театром. Театральний заклад, як комерційна установа зацікавлений в якомога більшій кількості клієнтів. Для ефективної взаємодії з існуючими та потенційними клієнтами застосовуються CRM-системи.

Customer Relationship Management - програмне забезпечення, яке використовується з метою автоматизації стратегій ефективної роботи з клієнтами[2]. Це робиться для збільшення рівня продажів, оптимізації маркетингу і поліпшення обслуговування клієнтів.

Основні функції CRM систем такі:

– Спрощують процес управління контактами, а все завдяки єдиній базі клієнтів.

– Пропонують точну картину взаємодії з клієнтами, що дає шанс проаналізувати їхні потреби на майбутнє.

– Керують потенційними і укладеними угодами. В результаті можна зробити певний прогноз роботи, уникнути конфліктів і проаналізувати роботу.

– Зберігають всю інформацію про клієнтів, співробітників, конкурентів і так далі. Все це підвищує компетентність співробітників.

– Допомагають планувати роботу з клієнтами з урахуванням практично всіх можливих нюансів.

– Генерують звіти за різними напрямками, завдяки чому можна планувати роботу компанії.

Сьогодні для того, щоб стати глядачем вистави, необхідно купити квиток. Купівля квитка відбувається в касі театру, онлайн на сайті театру або через онлайн-сервіс з продажу квитків, що надає послуги посередника.

Процес купівлі квитка можна інтегрувати з CRM-системою, що дозволить відразу створити запис контакта в системі і вести подальшу роботу з данним клієнтом.

Для більш зручного користування онлайн сервісами для купівлі квитків було впроваджено рекомендаційну систему.

Рекомендаційна система - програма, яка передбачає, які об'єкти (музичні композиції, фільми, книги) будуть цікаві користувачеві, грунтуючись на певну інформацію з профіля користувача[3].

Деякі рекомендаційні системи використовують комбінації підходів фільтрації вмісту та колаборотивної фільтрації — гібридний метод. Є кілька основних варіантів комбінування різних методів в рекомендаційних системах гібридного типу:

1. Побудова єдиної моделі, що використовує характеристики обох методів.

2. Впровадження item-based характеристик в user-based системи.

3. Впровадження user-based характеристик в item-based системи.

4. Впровадження item-based і user-based підходів окремо і використання комбінацій отриманих рейтингів.

Більш детально розглянемо перший метод, а саме «Побудова єдиної моделі, що використовує характеристики обох методів».

Фільтрація вмісту базується на створенні профілю користувача та об'єкта. Слід враховувати параметри об’єкта та їх відповідність вподобаннями користувача. Для цього в системах рекомендацій використовують теги для опису об'єктів, а профілю користувача відображає певну оцінку тегів або їх сукупності.

1.1.1 Опис процесу діяльності

В графічних метеріалах знаходиться схема структурна варіантів використань на якій зображено, які дії може виконувати користувач системи.

1.1.2 Опис функціональної моделі

Акторами платформи є:

– адміністратор;

– гість сайту;

– відвідувач;

– працівник театру.

Визначимо, які дії або варіанти використання вони виконують в комплексі задач, для цього наведемо таблицю 1.1, в якій описані актори, варіанти використання та їх описи дій.

Таблиця 1.1 – Опис дії варіантів використання

Актор	Назва варіанту	Опис	Пріоритет
Адміністратор	Адміністрування CRM-системи	Адміністратор додає до CRM-системи театральні вистави та супутні дані	Високий
Гість сайту	Персоніфікувати вікно бронювання	Гість сайту може вводити особисту інформацію у вікно бронювання театральної вистави	Високий
	Переглядати існуючі вистави	Гість сайту може зайти на сторінку вистави і продивитися всю наявну інформацію.	Середній
	Бронювати вистави	Гість сайту може забронювати квиток на виставу	Високий
Відвідувач	Отримувати СМС	Відвідувач може отримати СМС після бронювання	Високий

Продовження Таблиці 1.1

Актор	Назва варіанту	Опис	Пріоритет
Працівник театру	Переглядати інформацію про виставу	Актори можуть переглядати вистави в яких приймають участь, дивитися акторський склад	Високий
	Дивитися час репетицій	Актори можуть переглядати час репетицій	Середній

Відповідно визначених варіантів використання виявлено функціональні вимоги та встановлена їх пріоритетність, результат наведено у таблиці 1.2.

Таблиця 1.2 – Виявлені функціональні вимоги з варіантів використань

Актор	Варіант використання	Функціональна вимога	Пріоритет
Адміністратор	Наповнювати CRM-систему	1. Платформа має CRM-систему з усім інформаційним наповненням 1.1 Адміністратор слідкує за наповненням цієї системи	Високий Високий
Гість сайту	Персоніфікувати вікно бронювання	1. У вікні бронювання гість сайту заповнює особисті дані та натискає на кнопку «Забронювати» 2. Платформа бронює квиток на виставу	Високий Високий
	Переглядати існуючі вистави	1. Платформа надає можливості гостю сайту переглядати запропоновані вистави	Високий

Продовження Таблиці 1.2

Актор	**Варіант використання**	**Функціональна вимога**	**Пріоритет**
Гість сайту	Бронювати вистави	1. Платформа надає можливості бронювати виставу, що сподобалась гостю сайту	Високий
Відвідувач	Отримувати СМС	1. Відвідувач отримує повідомлення про успішне бронювання та список рекомендованих заходів	Високий
Працівник театру	Переглядати інформацію про виставу	1. Платформа надає можливості переглядати акторам інформацію про вистави, в яких вони грають	Високий
	Дивитися час репетицій	1. Платформа надає можливості переглядади акторам час репетицій	Високий

Огляд наявних аналогів

В ході пошуку схожих вирішень було виявлено деякі підходи зі схожою функціональністю:

– Афіша Карабас [4];

– Афіша Concert.ua[5];

– Афіша Yandex.ua[6];

– Афіша KudaGo[7].

В таблиці 1.3 наведено короткий опис характерних особливостей існуючих підходів.

Таблиця 1.3 – Характеристика існуючих підходів

Назва	Опис
Афіша Карабас	Веб-сайт та агентство з продажу квитків на розважальні заходи: концерти, шоу, театральні постановки, циркові вистави, фестивалі тощо. Метою створення даного сервісу було уникнення черг при купівлі квитків на розважальні заходи
Афіша Concert.ua	Concert.ua було створенно, щоб ділитися емоціями не лише під час подій, а й поза ними. Щоб користувачі не пропустили жодного концерту від улюблених зірок, змогли потрапити на важливу театральну прем'єру і встигли купити квитки на краще новорічне дійство для дітей. Також, щоб артисти відкривали себе глядачам, а організатори - створювали події, про які будуть говорити «хочемо ще».

Продовження Таблиці 1.3

Назва	Опис
Афіша Yandex.ua	Сервіс з продажу квитків на розважальні заходи: театри, концерти, кіно, мюзикли, шоу та інші події. Сервіс відкрився в 2005 році і містив розкладу кіно, театрів, концертів і виставок. У 2011 році з'явилася можливість купувати квитки в кіно. У 2015-му відбувся повний перезапуск: оновилися архітектура, дизайн, партнерська модель і стала доступна покупка квитків на концерти, спектаклі, шоу, мюзикли і дитячі події. У всіх рубриках сервісу працює формула ранжирування подій, яка вибудовує стрічку в певній послідовності. На позицію події в загальній стрічці впливає місткість майданчика, наявність квитків в онлайн-продажу, обсяг трафіку на сторінці та інші чинники.
Афіша KudaGo	На даному сайті є можливість купувати онлайн квитки в театр, кіно, квест-кімнати, музеї, виставки, Stand-Up шоу, курси з малювання, VR виставки тощо. Також даний сайт має можливість бронювати місці у екскурсіях.

Проаналізувавши програмні продукти, що вже існують, можна побачити, що на ринку України є онлайн сервіси для купівлі квитків на розважальні заходи. Проте немає сервісу лише для театральних вистав, а тим паче, немає сервісу, з рекомендаційною системою. Тому було вирішено

розробити комплекс задач з рекомендаційною системою, для купівлі квитків на театральні вистави.

1.2 Постановка задачі

1.2.1 Призначення розробки

Призначенням інформаційної системи є популяризація відвідування театральних заходів через забезпечення зручного процесу бронювання квитків на театральні вистави з рекомендаційною базою.

1.2.2 Цілі та задачі розробки

Цілями дипломного проекту є

– розробка зручного сервісу для бронювання квитків;

– надання рекомендацій користувачам театральних вистав.

Для досягнення поставленої мети мають бути вирішені такі задачі та реалізовані функції:

– бронювання квитків;

– перегляд вистав;

– створення рекомендацій театральних вистав, для користувача.

Висновок до розділу

В даному розділі описані предметне середовище та процес діяльності.

Представлено опис функціональної моделі, де зазначені актори, та дії, які вони виконують в комплексі задач. На основі визначених варіантів використання, виявлені функціональні вимоги.

Проведено пошук та аналіз існуючих аналогів платформи, сформульовані призначення, цілі та задачі розробки.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Вхідні дані рекомендаційної системи для театральних вистав надходять з декількох джерел, а саме від:

– адміністратора;

– гостя сайту;

– актора.

Тепер детально розглянемо, які саме дані надходять з цих джерел.

Дані, які надходять від глядачів.

При бронюванні місць на виставу на платформі, глядач повинен повідомити наступні дані:

– ПІБ;

– контактний номер телефону.

Дані, які надходять від адміністратора.

При створенні вистави, адміністратор має повідомити наступні дані:

1. Для глядача
   – місце, де буде проходити вистава;
   – назва вистави;
   – час початку вистави.
2. Для актора
   – місце, де буде проходити репетиція;
   – назва вистави;
   – час початку та кінця репетиції;
   – перелік акторів, що беруть учать у виставі.

Дані, які надходять від акторів.

Актори, що користуються даною платформою повідомляють наступні дані:

– ПІБ;
– номер телефону;
– електрону адресу.

2.2 Вихідні дані

Вихідними документами є форми створені в CRM-системі, на яких відображається основна інформація щодо вистав, розклад в Outlook та СМС повідомлення.

Приклади вихідних даних, зображені на рисунках 2.1-2.9.

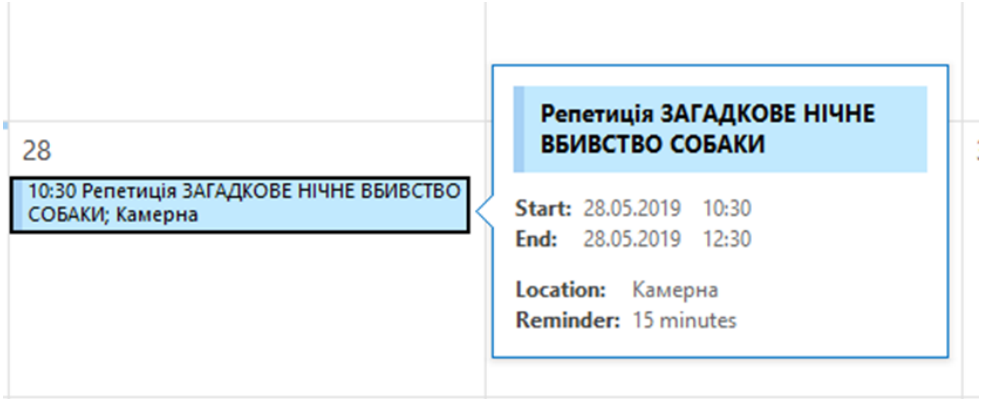


Рисунок 2.1 – Інформація про репетицію вистави для учасників в Outlook

Рисунок 2.2 – Інформація про репетицію вистави в CRM

Рисунок 2.3 – Інформація про захід в CRM

Рисунок 2.4 – Інформація про сцену в CRM

Рисунок 2.5 – Інформація про виставу в CRM

Рисунок 2.6 – Інформація про жанр в CRM

Рисунок 2.7 – Інформація про контакт в CRM

Рисунок 2.8 – Інформація про роль контакта в CRM

Рисунок 2.9 – СМС повідомлення про бронювання

2.3 Опис структури бази даних

У таблицях 2.1 – 2.7 наведена структура таблиць CRM-системи.

Таблиця 2.1 – Опис таблиці Booking Model

Код	Опис	Тип даних	Обов'язкове	Унікальне	Клас
eventId	Ідентифікатор заходу	string	X	X	BookingModel
phone	Номер телефону	string	X	X	BookingModel
name	Прізвище та ім’я відвідувача	string	X	-	BookingModel

Таблиця 2.2 – Опис таблиці ContactModel

Код	Опис	Тип даних	Обов'язкове	Унікальне	Клас
FullName	Прізвище та ім’я контакта	string	X	-	ContactModel
MobilePhone	Номер телефону	string	X	X	ContactModel
PrimaryId	Ідентифікатор контакта	Guid	X	X	ContactModel
Role	Ідентифікатор ролі	Guid	X	X	RoleModel
Description	Опис	string	X	-	ContactModel

Таблиця 2.3 – Опис таблиці EventModel

Код	Опис	Тип даних	Обов'язкове	Унікальне	Клас
PrimaryId	Ідентифікатор заходу	Guid	X	X	EventModel
Spectacle	Ідентифікатор вистави	Guid	X	X	SpectacleModel
Date	Дата проведення заходу	Date and Time	X	-	EventModel
Name	Назва заходу	string	X	-	EventModel
Scene	Ідентифікатор сцени	Guid	X	X	SceneModel

Таблиця 2.4 – Опис таблиці SpectacleModel

Код	Опис	Тип даних	Обов'язкове	Унікальне	Клас
PrimaryId	Ідентифікатор вистави	Guid	X	X	SpectacleModel
Autor	Ідентифікатор контакта	Guid	X	X	ContactModel
Name	Назва вистави	string	X	X	SpectacleModel
Director	Ідентифікатор контакта	Guid	X	X	ContactModel

Таблиця 2.5 – Опис таблиці GenreModel

Код	Опис	Тип даних	Обов'язкове	Унікальне	Клас
PrimaryId	Ідентифікатор вистави	Guid	X	X	GenreModel
Name	Назва жанру	string	X	X	GenreModel

Таблиця 2.6 – Опис таблиці SceneModel

Код	Опис	Тип даних	Обов'язкове	Унікальне	Клас
PrimaryId	Ідентифікатор вистави	Guid	X	X	SceneModel
Name	Назва сцени	string	X	X	SceneModel

Таблиця 2.7 – Опис таблиці RoleModel

Код	Опис	Тип даних	Обов'язкове	Унікальне	Клас
PrimaryId	Ідентифікатор вистави	Guid	X	X	RoleModel
Name	Назва ролі	string	X	X	RoleModel

Висновок до розділу

В даному розділі описано вхідні дані, що надходять від відвідувачів та адміністратора. Основні вихідні дані виводяться в CRM-систему, а для максимально простого інформування глядача було реалізовано надсилання СМС повідомлення. Також була представлена структура таблиць збереження даних в CRM-системі.

3 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

Дипломний проект являє собою інтеграцію декількох підсистем. CRM система виступає, як базова платформа для зберігання та управління даними. Для реалізаціїї можливості бронювання місць на заходи було розроблено сервіс, що створює, в разі відсутності, нову картку контакта в ролі відвідувача в CRM системі, та зв’язує контакт та захід. Був розроблений односторінковий сайт з афішею та формою бронювання місць на захід. Для взаємодії сайту та сервісу, було реалізовано API, що приймає запити з сайту та передає дані до сервісу.

Усі використані технології опишемо окремо для кожної з частин.

Dynamics 365 for Sales

Пакет програмного забезпечення для управління взаємовідносинами з клієнтами, розроблений компанією Microsoft та орієнтований на організацію продажів, маркетингу та надання послуг. Dynamics CRM є клієнт-серверним додатком, що підтримує браузери Chrome, Firefox та Opera[8]. Пакет використовується, як розширена платформа для взаємодії з клієнтами, і може налаштовуватися залежно від цілей за допомогою програмної платформи .NET. Система надає механізм для налаштувань бізнес-логіки. Розробники можуть створювати її не лише через зміну процесів робочого циклу а також і через інтеграцію сторонніх сервісів з Microsoft Dynamics CRM, що і будуть виконувати певну логіку у відповідь на системні події в рамках певної структурної одиниці.

API

Для інтеграціїї сайту з CRM системою було розроблено ASP.Net Core Web Application. API (Application Programming Interface) - це набір функцій і правил, що дозволяє взаємодіяти з програмним забезпеченням, що надається API та іншими програмними компонентами[9]. У веб-розробці під API

зазвичай розуміється набір стандартних методів, властивостей, подій і URL-посилань для взаємодії з веб-контентом. Під час реалізації рішення було використано фреймворк .Net Framework 4.6.2 та бібліотеку Microsoft.Xrm.Sdk.

Microsoft .NET — програмна технологія, запропонована фірмою Microsoft, як платформа для створення, як звичайних програм, так і веб-застосунків[10]. Багато в чому є продовженням ідей та принципів, покладених в технологію Java. Однією з ідей .NET є сумісність служб, написаних різними мовами. Хоча ця можливість рекламується Microsoft як перевага .NET, платформа Java має таку саму можливість.

.NET — крос-платформова технологія, в цей час існує реалізація для платформи Microsoft Windows, FreeBSD (від Microsoft) і варіант технології для ОС Linux в проекті Mono (в рамках угоди між Microsoft з Novell), DotGNU.

.NET поділяється на дві основні частини — середовище виконання (по суті віртуальна машина) та інструментарій розробки.

Як і технологія Java, середовище розробки .NET створює байт-код, призначений для виконання віртуальною машиною. Вхідна мова цієї машини в .NET називається CIL (Common Intermediate Language), також відома як MSIL (Microsoft Intermediate Language), або просто IL. Застосування байт-коду дозволяє отримати крос-платформність на рівні скомпільованого проекту (в термінах .NET: збірка), а не на рівні вихідного тексту, як, наприклад, в C. Перед запуском збірки в середовищі виконання (CLR) байт-код перетворюється вбудованим в середовище JIT-компілятором (just in time, компіляція на льоту) в машинні коди цільового процесора.

Microsoft.Xrm.Sdk містить інформацію та ресурси для розробників, які створюють розширення на основі коду для Dynamics 365.

SDK включає в себе архітектурний огляд програми Dynamics 365, модель об'єктної сутності, модель безпеки та веб-служби. Зразок коду та покрокові інструкції надаються для ознайомлення з новими функціями. Він також

містить інформацію для розробників, які налаштовують веб-клієнт або Dynamics 365 для Microsoft Office Outlook, включаючи сценарії, інтеграцію користувацьких веб-сторінок і зразки коду.

На додаток до документації, цей пакет завантаження містить збірки та засоби, необхідні для розробки, допоміжний код для аутентифікації, і проекти Microsoft Visual Studio для зразка коду, що міститься в документації.

Клієнтська частина

Flask - це основа для створення веб-додатків на мові програмування Python за допомогою інструментарію Werkzeug, а також шаблону Jinja2[11]. Відноситься до категорії так званих microframeworks - мінімалістської рамки веб-додатків, які свідомо забезпечують лише найбільш основні можливості.

Python - популярна мова програмування високого рівня, призначена для створення різноманітних програм. Це веб-програми, ігри та настільні програми, а також робота з базами даних. Python отримав досить великий розподіл в області машинного навчання і дослідження штучного інтелекту.

Основні можливості мови програмування Python:

– мова сценаріїв, код програми визначається як сценарії;

– підтримка найрізноманітніших парадигми програмування, включаючи об'єктно-орієнтовані та функціональні парадигми;

– інтерпретація програм, lля роботи зі скриптами потрібен інтерпретатор, який запускає і виконує сценарій;

– автоматичне керування пам'яттю;

– динамічне введення тексту.

Виконання програми на Python виглядає так. Спочатку ми пишемо сценарій з набором виразів у заданій мові програмування в текстовому редакторі. Ми передаємо цей сценарій інтерпретатору. Інтерпретатор переводить код у проміжний байт-код, а потім віртуальна машина переводить отриманий байтовий код у набір інструкцій, які виконуються операційною системою.

Тут варто відзначити, що хоча формальне переведення вихідного коду в байт-код і переведення байткода віртуальною машиною в набір машинних команд представляють два різних процеси, але насправді вони об'єднуються в самому перекладачі.

HTML (мова гіпертекстової розмітки) — це код, який використовується для структурування і відображення веб-сторінки та її контенту. Наприклад, контент може бути розбитий на параграфи (абзаци), містити список, зображення чи таблицю.

HTML не є мовою програмування, це мова розмітки, яка говорить вашому браузеру, як відображати вміст веб-сторінки, яку ви переглядаєте. Вона може бути простою або складною, залежно від бажання дизайнера її створити. HTML складається з серії (елементи), які використовуються для розміщення або "обтікання" у них різних частин вмісту таким чином, щоб вони з'являлися або діяли певним чином. Ці елементи, використовуючи як початковий, так і кінцевий теги, можуть пов'язувати слово або зображення з будь-якою іншою сторінкою, відображатися курсивом, збільшувати або зменшувати шрифт тощо.

CSS (англ. Cascading Style Sheets - каскадні таблиці стилей) - формальний мовний опис зовнішнього вигляду документа, написаного з використанням мови розмітки.

CSS використовується дизайнерами веб-сторінок для визначення кольорів, шрифтів, розташування окремих блоків та інших аспектів, що відображають зовнішній вигляд цих веб-сторінок. Основною метою розробки CSS було відокремлення опису логічної структури веб-сторінки (яка виконується за допомогою HTML або інших мов розмітки) з опису зовнішнього вигляду цієї веб-сторінки (яка тепер виконується з використанням формального мови CSS).). Таке поділ може збільшити доступність документа, забезпечити більшу гнучкість і контроль за його поданням, а також зменшити складність і повторюваність структурованого

контенту. Крім того, CSS дозволяє подавати той самий документ в різних стилях або методах виводу, таких як перегляд екрану, друк, читання голосу (спеціальний голосовий браузер або екранний пристрій), або при виведенні пристроїв за допомогою шрифту Брайля.

Node.js - це програмна платформа, заснована на движку V8 (переведення JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованої мови на мову загального призначення. Node.js додає можливість JavaScript для взаємодії з пристроями вводу-виводу через свій API (написаний на C ++), для підключення інших зовнішніх бібліотек, написаних на різних мовах, забезпечуючи виклики до них з коду JavaScript. Node.js використовується в основному на сервері, виступаючи в ролі веб-сервера, але можна розробляти настільні програми на Node.js і навіть програмні мікроконтролери. Node.js заснований на орієнтованому на події та асинхронному (або реактивному) програмуванні з неблокуючим введенням-виведенням.

Розгортання продукту

Як середовище для розгортання API була обрана платформа Azure, оскільки це відкрита та гнучка хмарна платформа, що дозволяє швидко створювати, розгортати та управляти додатками на глобальній мережі центрів обробки даних Microsoft.

Для розгортання клієнтської частини було обрано Docker- програмне забезпечення для автоматизації розгортання і управління додатками в операційних системах з підтримкою контейнеризації.

3.2 Вимоги до технічного забезпечення

3.2.1 Загальні вимоги

Для коректного функціонування цієї програми технічні засоби повинні відповідати наступним вимогам:

– комп’ютер з такою конфігурацією:

1) процесор з тактовою частотою не нижче 1 ГГц;

2) об’єм RAM (не менше 256 МБ);

3) до інших складових вимоги не висуваються, оскільки вони впливають на роботу програми несуттєво;

– додатково має бути встановлене таке програмне забезпечення:

1) операційна система Windows/Linux/OSX;

2) будь-який браузер, що підтримує стандарти ES6, наприклад Google Chrome, Opera, Mozilla Firefox;

– комп’ютерна периферія, така як:

1) монітор;

2) мишка;

3) клавіатура.

3.3 Архітектура програмного забезпечення

3.3.1 Діаграма класів

Схема структурна класів представлена у частині графічного матеріалу.

Діаграма містить 15 класів, а саме:

BookingModel- клас, що описує структуру даних, які приходять до контролеру у вигляді http-запиту, та проводить валідацію номеру тлефону.

EntityBase- цей клас представляє базовий набір параметрів та методів для CRUD операцій до CRM-системи. Даний клас є батьківським для усіх класів описів моделей даних.

ContactModel- клас-нащадок від EntityBase, що описує структуру даних контакту. Зберігає в собі інформацію про імя та прізвище контакта, його номер телефону та роль.

EventModel- клас-нащадок від EntityBase, що описує структуру даних заходу. Зберігає в собі інформацію про виставу, дату, час та місце проведення. А також містить список відвідувачів, що забронювали місце.

GenreModel- клас-нащадок від EntityBase, що описує структуру даних жанру. Зберігає в собі інформацію про назву жанру. Вистава може бути характеризована кількома жанрами.

SpectacleModel- клас-нащадок від EntityBase, що описує структуру даних вистави. Зберігає в собі інформацію про назву вистави, автора твору та режисера. Також містить список акторів, що задіяні в виставі.

Contact- клас-нащадок від ContactModel, що містить набір конструкторів, для створення провайдера, що реалізує методи для CRUD операцій до CRM-системи, розширений полями ContactModel.

Event- клас-нащадок від EventModel, що містить набір конструкторів, для створення провайдера, що реалізує методи для CRUD операцій до CRM-системи, розширений полями EventModel.

Genre- клас-нащадок від GenreModel, що містить набір конструкторів, для створення провайдера, що реалізує методи для CRUD операцій до CRM-системи, розширений полями GenreModel.

Spectacle- клас-нащадок від SpectacleModel, що містить набір конструкторів, для створення провайдера, що реалізує методи для CRUD операцій до CRM-системи, розширений полями SpectacleModel.

BaseCrmController- батьківський клас, що реалізує доступ до:

– IOrganizationService- сервіс доступу до CRM;
– ISpectacleService- сервіс обробки, отриманих в запиті, даних;
– ILogger- сервіс логування.

SpectacleController- клас-нащадок від BaseCrmController, що реалізує методи прийому http-запитів, та відправку даних до сервісів для подальшої обробки.

SpectacleService- клас, котрий реалізує логіку створення контактів, бронювання місця та підбору рекомендованих заходів.

3.3.2 Діаграма послідовності

Схема структурна послідовності дій наведена у розділі графічного матеріалу. На діаграмі відображена послідовність дій, виконуваних гостем сайту, відвідувачем, адміністратором CRM та працівником театру.

3.3.3 Діаграма компонентів

Схема структурна компонентів зображена у розділі графічного матеріалу.

На даній діаграмі відображені компоненти, що використовуються в комплексі задач, та взаємозв'язки між ними. Основними компонентами в системі є відвідувач, веб-сайт, веб-інтерфейс, сервіс та система CRM. Відвідувачі використовують веб-сайт, що при бронюванні відправляє запит на веб-інтерфейс, який у свою чергу передає дані на обробку сервісу, що відповідає за інтеграцію з CRM-системою.

3.3.4 Специфікація функцій

Функції класів програмного забезпечення наведені в таблиці 3.1.

Таблиця 3.1 – Функції класу AdminService програмного забезпечення

Функція	Опис функції
`void Book(BookingModel bookingDetail, IOrganizationService service)`	Призначена для перевірки наявності контакту в CRM, та створенні такого в разі відсутності.
`List<string> GetRecomendation(Event currentEvent, IOrganizationService service)`	Призначена для повернення сформованого списку рекомендованих заходів.
`Dictionary<Guid, int> GetRating(List<string> currentSpectacleTags, Dictionary<Guid, List<string>> spectaclesTags)`	Призначена для визначення рейтингу рекомендованості заходів.

Продовження Таблиці 3.1

Функція	**Опис функції**
`Dictionary<Guid, List<string>> GetTags(Event currentEvent, IOrganizationService service)`	Призначена для визначення тегів відвіданих заходів.
`List<Spectacle> getVisitorSpectacles(Contact visitor, IOrganizationService service)`	Призначена для повернення переліку відвіданих контактом заходів.
`List<string> GetSpectacleTags(Guid id, IOrganizationService service)`	Призначена для визначення тегів поточного заходу.
`string GetActors(Guid id, IOrganizationService service)`	Призначена для повернення переліку акторів, що беруть участь в заході.
`string GetGeneres(Guid id, IOrganizationService service)`	Призначена для повернення переліку жанрів, що характеризують захід.
`List<Contact> GetVisitors(Guid id, IOrganizationService service)`	Призначена для повернення переліку відвідувачів заходу.

Висновок до розділу

Цей розділ описує використані інструменти розробки для реалізації програмного забезпечення системи підбору театральних заходів.

Було обгрунтовано вибір засобів розробки .Net, Phyton і Node.JS

Архітектура програмного забезпечення в цьому розділі була представлена діаграмами класів, послідовностей і компонентів.

Надана специфікація функцій, що реалізують процес підбору рекомендованих театральних заходів.

4 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Змістовна постановка задачі

Робота театру повязана з такими сутностями як вистава, захід та користувач. Опишемо кожну з сутностей детальніше.

Вистава характеризується жанрами, автором, режисером та акторами. Кожен з параметрів характеристики є тегом, що описує виставу, в сукупності формуючи словник тегів. Для конкретної вистави такий словник містить значення 1 для тегу, якщо тег присутній в описі вистави та 0, якщо ні.

Захід характеризується виставою, датою та місцем проведення та списком відвідувачів.

Відвідувач містить особисті дані та історію відвіданих заходів.

Коли гість сайту бронює місце на захід на основі історій відвідувачів, що уже відвідали дану виставу, складається усереднений профіль відвідувача. На основі усередненого профілю підібрати, щонайцікавіші для нового відвідувача події.

4.2 Математична постановка задачі

Рекомендаційна система - програма, яка намагається передбачити, на основі конкретної інформації історію відвідувань користувача, які вистави представляють інтерес для користувача.

Формально завданням може бути пошук рекомендованої вистави описаний наступним чином:

$$\forall u \in U,\ s'_u = arg\ max_{s \in S}\ h(u, s), \tag{4.1}$$

де U - набір відвідувачів, S - набір вистав, які можуть бути рекомендованими для відвідувачів, h - функція, яка визначає, наскільки деякі вистави відповідають деяким відвідувачам.

Тому необхідно вибрати таку виставу $s' \in S$, для якої значення задоволеності відівдувача $u \in U$ є максимальним.

4.3 Обґрунтування методу розв’язання

Задача (4.1) представляє собою задачу фільтрації вмісту через визначення задоволення користувача *u* певною виставою *s*. Методом розв’язання задачі є TF-IDF[12].

TF-IDF- це числова статистика, призначена для відображення того, наскільки важливим є слово для об’єкта. Його часто використовують, як ваговий коефіцієнт у пошуку інформації, видобутку тексту та моделюванні користувачів. Значення TF-IDF зростає пропорційно кількості разів, коли слово з'являється в об’єкті і компенсується величиною, зворотною частоті входження тега в об'єкт колекції. Це допомагає зробити деякі слова більш загальними. TF-IDF є однією з найпопулярніших схем зважування термінів сьогодні; 83% текстових рекомендаційних систем у цифрових бібліотеках використовують TF-IDF.

4.4 Опис методу розвязання

У системах рекомендації з фільтрацією вмісту функція задоволення $h(u, s)$ користувача *u* за деякими виставами s визначається на основі інформації про задоволеність користувачів виставами $s_i \in S$, які схожі на *s*. Певним чином, історія відвідувача в системі формується з вистав, відвіданих користувачем, як набір параметрів, що визначають виставу *s*. Такі параметри зазвичай є ключовими словами та відповідними вагами для кожного об'єкта. Тому необхідно визначити ваги цих параметрів. У пошуку інформації одним з найбільш відомих методів визначення ваг ключових слів є показник TF-IDF.

Припустимо, що N - загальна кількість об'єктів, які можуть бути рекомендовані користувачеві, а також, що тег k_j характеризує виставу n_i, а $f_{i,j}$ - кількість входжень цього тегу до історії d_j. Тоді $TF_{i,j}$ (term frequency) - це відношення числа входжень тега до загальної кількості тегів історії відвіданих вистав, тобто:

$$TF_{i,j} = \frac{f_{i,j}}{max_z f_{z,j}} \quad (4.2)$$

Але якщо враховувати тільки частоту входження тега, то в більшості історій відвідуваня максимальна вага буде у найпоширеніших тегів, що, найімовірніше, призведе до неправильного оцінювання переваг користувача. Для уникнення подібного застосовується IDF_i (inverse document frequency) - величина, зворотна частоті входження тега до історії користувача. Визначаємо її як:

$$\mathrm{IDF}_i = log \frac{N}{n_i} \quad (4.3)$$

Таким чином, вага $w_{i,j}$ у тега k_i в об'єкті d_j позначається, як добуток частоти входження тега на зворотну частоту:

$$w_{i,j} = TF_{i,j} \times IDF_i \quad (4.4)$$

В такому випадку, контент об'єкта d_j визначаємо як:

$$Content\left(d_{j,i}\right) = (w_{1,j}, \ldots, w_{k,j}) \quad (4.5)$$

Як зазначалося вище, рекомендаційні системи з фільтрацією вмісту пропонують вистави з урахуванням тих, які користувач відвідав раніше. Різні вистави порівнюються з тими, що були відвідані користувачем і з них рекомендуються ті, які мають максимальну схожість. Сукупність вистав, які користувач відвідав раніше, утворює визначену для користувача історію *ContentBasedProfile(u)* або, інакше кажучи, вектор ваг ($w_{u,1}$, ..., $w_{u,k}$), де кожна вага $w_{u,i}$ визначає важливість тега k_i для користувача u. Отже, *ContentBasedProfile(u)* і *Content(s)* можна уявити як TF-IDF вектори $\vec{w}_u$ і $\vec{w}_s$, при цьому функція задоволеності користувача *h(u,s)* може бути представлена як косинусний коефіцієнт векторів $\vec{w}_u$ і $\vec{w}_s$:

$$h(u,s) = \cos(\vec{w}_u, \vec{w}_s) = \frac{\vec{w}_u \vec{w}_s}{\|\vec{w}_u\| \|\vec{w}_s\|} = \frac{\sum_{i=1}^{K} \vec{w}_{i,u} \vec{w}_{i,s}}{\sqrt{\sum_{i=1}^{K} w^2{}_{i,u}} \sqrt{\sum_{i=1}^{K} w^2{}_{i,s}}} \quad (4.6)$$

, де K - загальна кількість тегів в системі. Серед отриманих оцінок схожості обираються 3 з найбільшим значенням та додаються до СМС-повідомлення.

Наведемо алгоритм підбору театральних заходів:

КРОК 1. З запиту бронювання місця, що приходить до контролера API з сайту від потенційного відвідувача отримати дані про виставу, що грається на театральній події.

КРОК 2. Отримати список відвідувачів, які були на заходах, де гралася ця ж вистава та історії їх відвідувань.

КРОК 3. Провести агрегацію історій відвідувачів з метою отримання агрегованого відвідувача з сумарною історією.

КРОК 4. Визначити вектор ваг тегів для агрегованого відвідувача.

КРОК 5. Розрахувати вектор задоволеності агрегованого користувача виставами.

КРОК 6. Обрати 3 вистави з найбільшим значенням функції задоволеності.

Висновок до розділу

У цьому розділі визначено змістовне і математичне формулювання завдання підбору театральних подій на основі переваг усередненого відвідувача поточного заходу.

Описано та обгрунтовано вибір методу вирішення поставленого завдання.

Детально описано алгоритм отримання даних та самого підбору рекомендованих театральних подій.

Надано докладний опис математичних розрахунків, що ведуться під час роботи алгоритмів, що використовуються для вирішення поставленої задачі.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

5.1.1 Керівництво Адміністратора

Для початку роботи з системою необхідно перейти за посиланням https://adventtheatre.crm4.dynamics.com, після чого система запросить дані для входу (рисунок 5.1-5.2)

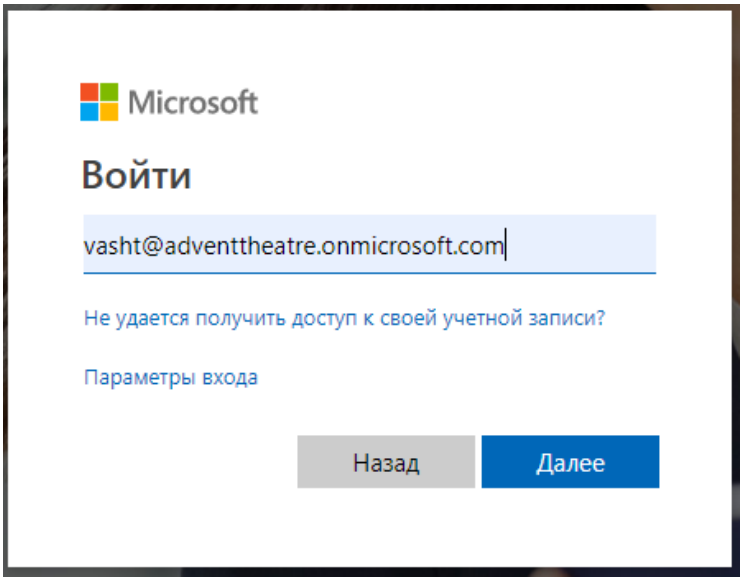


Рисунок 5.1 – Форма введення логіна

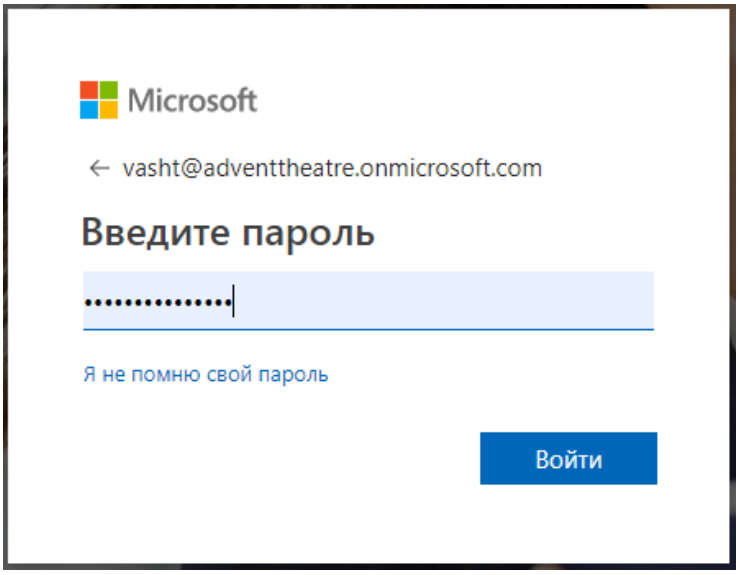


Рисунок 5.2 – Форма введення паролю

Після авторизації Адміністратор має можливість відкрити в меню розділ «Театр», де можна обрати один з пунктів управління(рисунок 5.3).

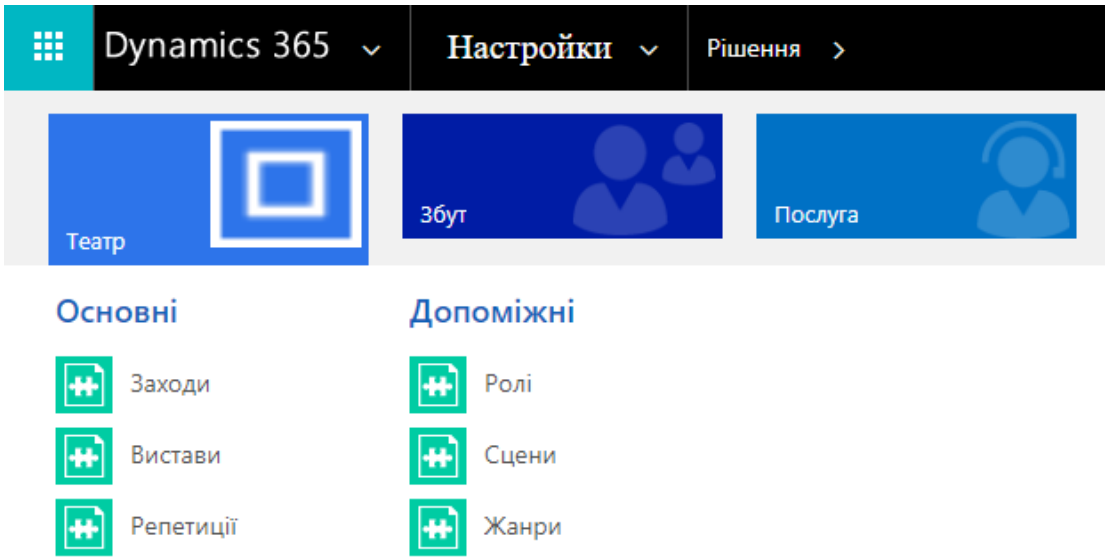


Рисунок 5.3 – Меню управління системою

Для управління ролями контактів, необхідно обрати вкладку Ролі, після чого будуть відображені усі активні записи Ролі(рисунок 5.4).

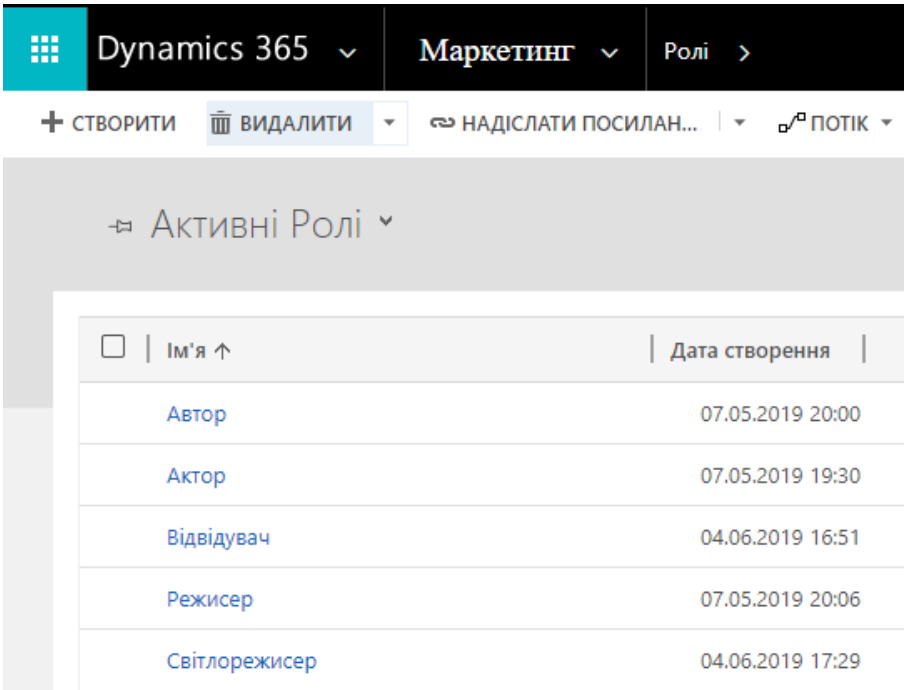


Рисунок 5.4 – Форма з активними Ролями

Для створення нового запису натискаємо Створити, відкривається сторінка де Адміністратор має можливість ввести дані та зберегти запис натисканням на Зберегти(рисунок 5.5).

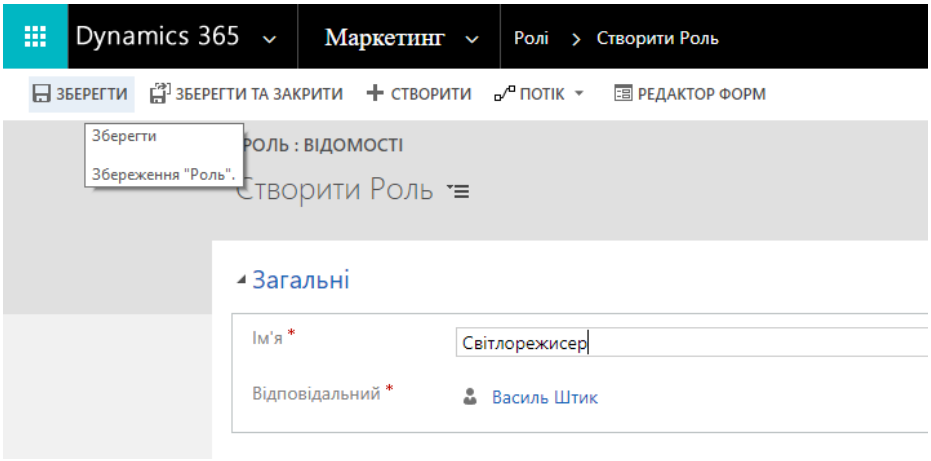


Рисунок 5.5 – Форма створення Ролі

Для управління сценами, необхідно обрати вкладку Сцени, після чого будуть відображені усі активні записи Сцени(рисунок 5.6).

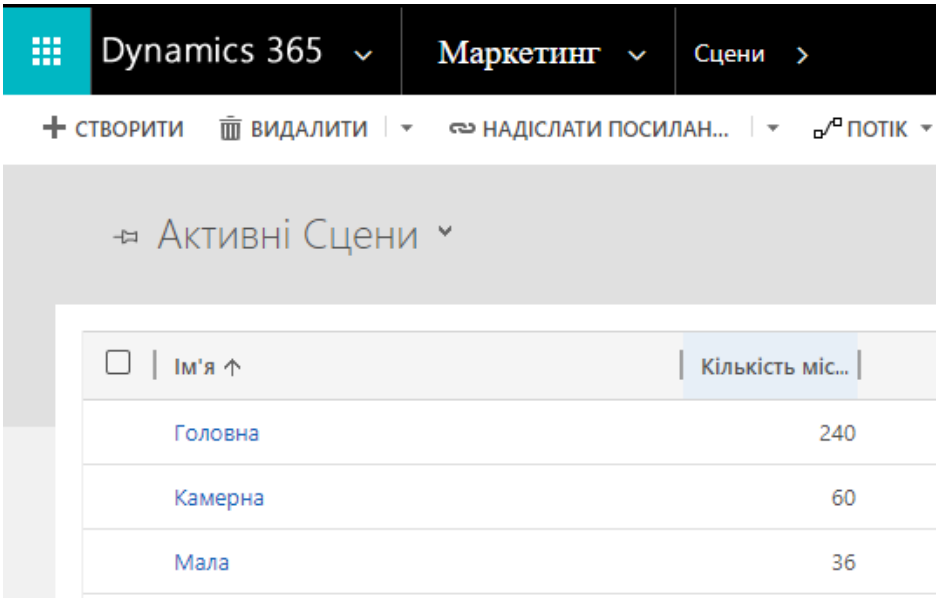


Рисунок 5.6 – Форма з активними Сценами

Для створення нового запису натискаємо Створити, відкривається сторінка де Адміністратор має можливість ввести дані та зберегти запис натисканням на Зберегти(рисунок 5.7).

Рисунок 5.7 – Форма створення Сцени

Для управління жанрами вистав, необхідно обрати вкладку Жанри, після чого будуть відображені усі активні записи Жанри(рисунок 5.8).

Рисунок 5.8 – Форма з активними Жанрами

Для створення нового запису натискаємо Створити, відкривається сторінка де Адміністратор має можливість ввести дані та зберегти запис натисканням на Зберегти(рисунок 5.9).

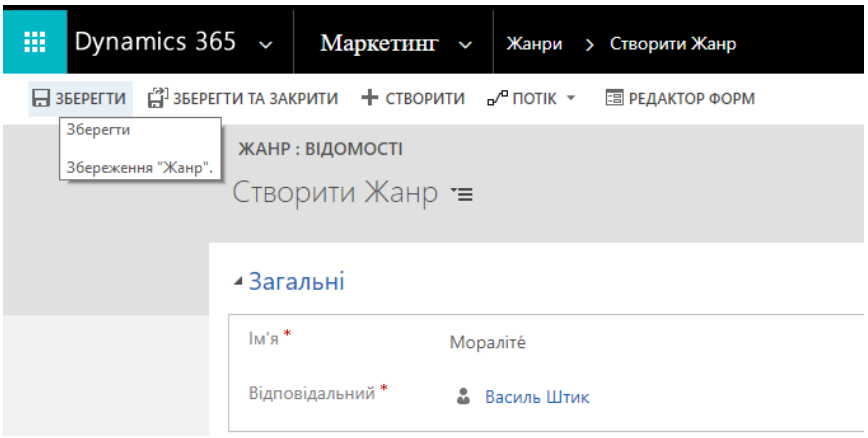


Рисунок 5.9 – Форма створення Жанру

Для управління виставами, необхідно обрати вкладку Вистави, після чого будуть відображені усі активні записи Вистави(рисунок 5.10).

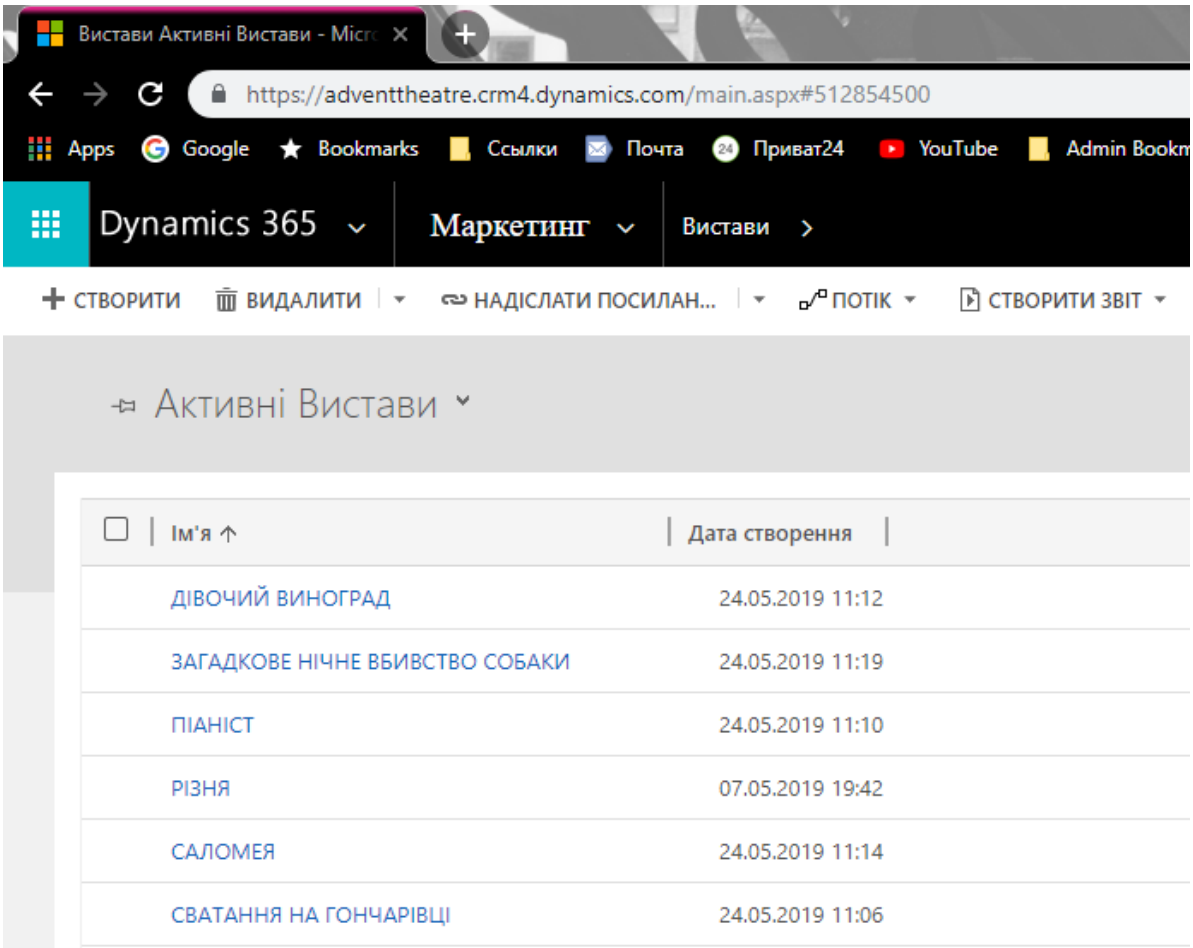


Рисунок 5.10 – Форма з активними Виставами

Для створення нового запису натискаємо Створити, відкривається сторінка де Адміністратор має можливість ввести дані та зберегти запис натисканням на Зберегти(рисунок 5.11).

Рисунок 5.11 – Форма створення Вистави

Після збереження є можливість додати Акторів та Жанри, для чого необхідно натиснути на «+» на відповідній формі(рисунок 5.12-5.13).

Рисунок 5.12 – Форма додавання Акторів

Рисунок 5.13 – Форма додавання Жанрів

Для перегляду запису натискаємо на відповідний рядок, відкривається сторінка, де Адміністратор має можливість преглянути та внести правки до запису. Для збереження внесених змін необхідно натиснути на значок дискети в правому нижньому куті вікна(рисунок 5.14).

Рисунок 5.14 – Форма запису Вистави

Для управління заходами, необхідно обрати вкладку Заходи, після чого будуть відображені усі активні записи Заходи(рисунок 5.15).

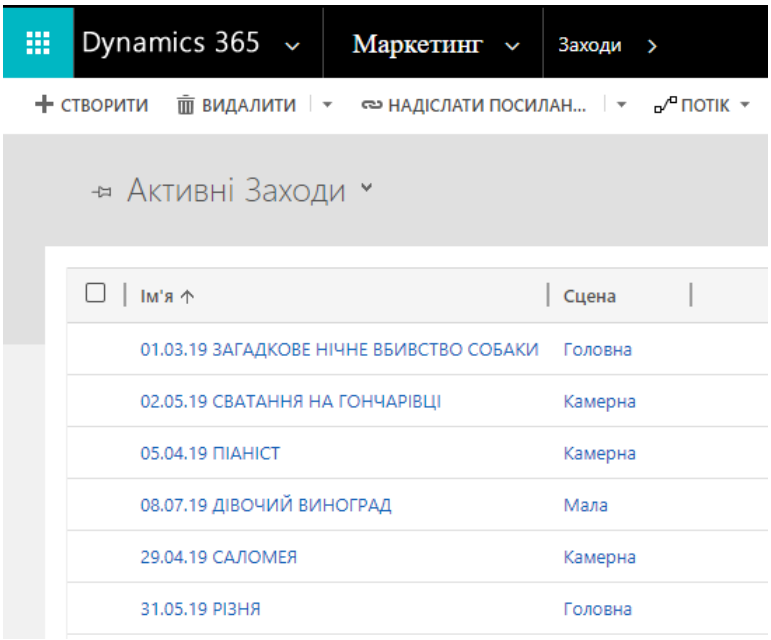


Рисунок 5.15 – Форма з активними Заходами

Для створення нового запису натискаємо Створити, відкривається сторінка де Адміністратор має можливість ввести дані та зберегти запис натисканням на Зберегти(рисунок 5.16).

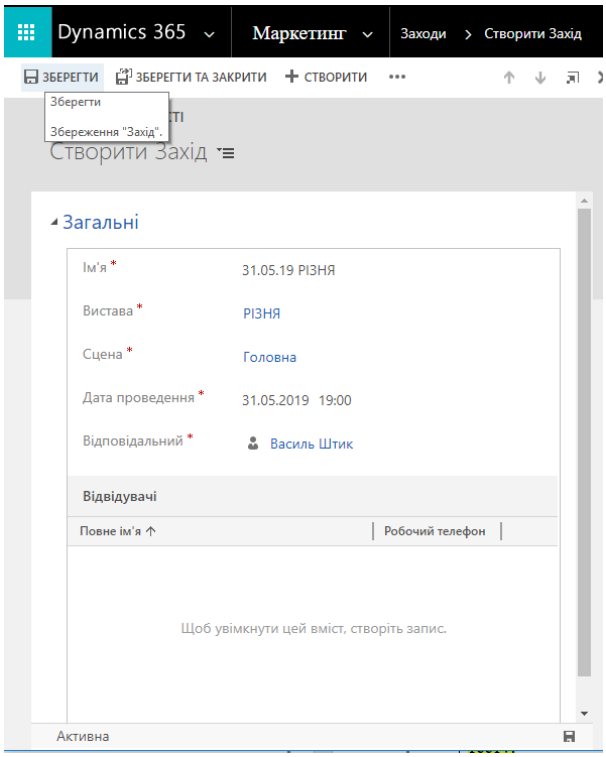


Рисунок 5.16 – Форма створення Заходу

Для перегляду запису натискаємо на відповідний рядок, відкривається сторінка де Адміністратор має можливість преглянути та внести правки до запису. Для збереження внесених змін необхідно натиснути на значок дискети в правому нижньому куті вікна(рисунок 5.17).

Рисунок 5.17 – Форма запису Заходу

Для управління репетиціями, необхідно обрати вкладку Репетиції, після чого будуть відображені усі активні записи Репетиції(рисунок 5.18).

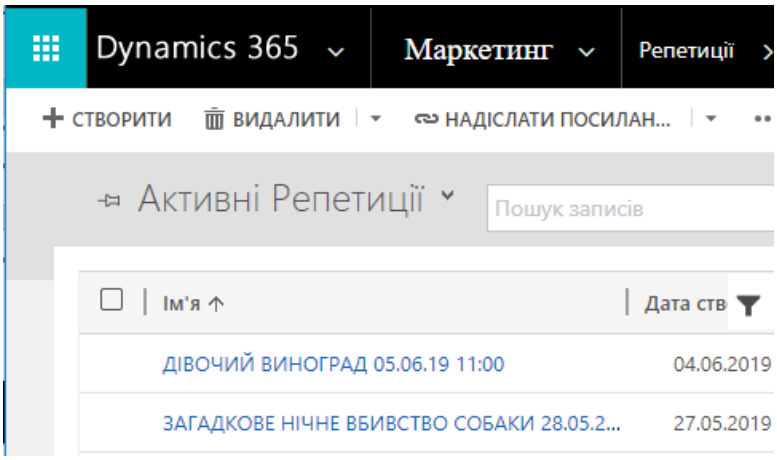


Рисунок 5.18 – Форма з активними Репетиціями

Для створення нового запису натискаємо Створити, відкривається сторінка де Адміністратор має можливість ввести дані та зберегти запис натисканням на Зберегти(рисунок 5.19).

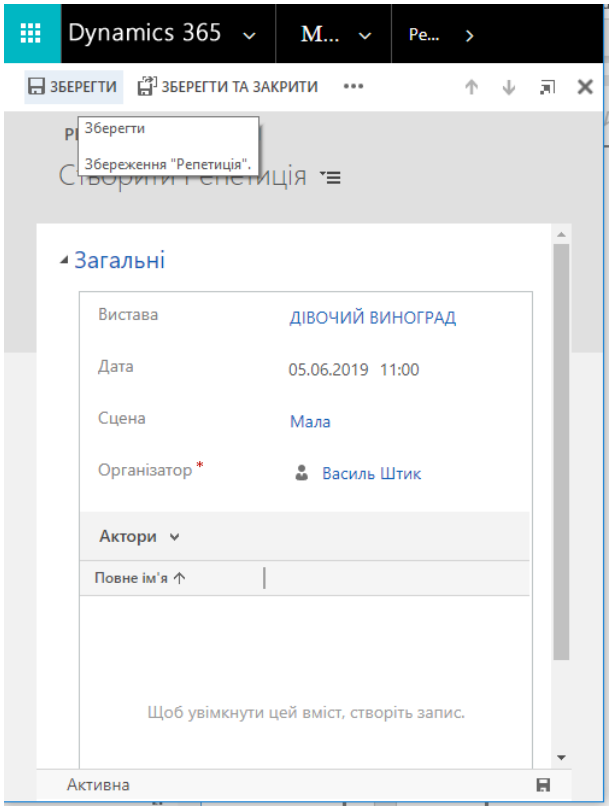


Рисунок 5.19 – Форма створення Репетиції

Для перегляду запису натискаємо на відповідний рядок, відкривається сторінка, де Адміністратор має можливість преглянути та внести правки до запису. Для збереження внесених змін необхідно натиснути на значок дискети в правому нижньому куті вікна(рисунок 5.20).

Рисунок 5.20 – Форма запису Репетиції

Для управління контактами, необхідно обрати вкладку Контакти, після чого будуть відображені усі активні записи Контакти(рисунок 5.21).

Рисунок 5.21 – Форма з активними Контактами

Для створення нового запису натискаємо Створити, відкривається сторінка де Адміністратор має можливість ввести дані та зберегти запис натисканням на Зберегти(рисунок 5.22).

Рисунок 5.22 – Форма створення Контакта

5.1.2 Керівництво Працівника театру

Працівник театру зайшовши в календар свого облікового запису Outlook, бачить усі заплановані для нього Адміністратором заходи та репетиції(рисунок 5.23).

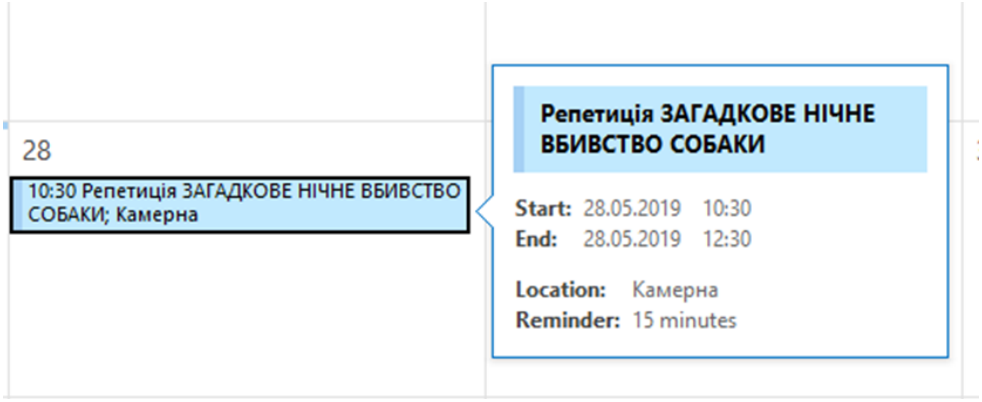


Рисунок 5.23 – Запланована репетиція для працівника театру

5.1.3 Керівництво Відвідувача

Для бронювання місць на захід необхідно перейти за посиланням https://adventtheatre.com, після чого буде вікрито сторінку з афішею(рисунок 5.24).

Рисунок 5.24 – Головна сторінка сайту з афішею

Гостю сайту пропонується обрати одну з доступних вистав натиснувши на неї, після чого буде відкрита сторінка з формою бронювання, де гість сайту вводить дані, необхідні для бронювання(рисунок 5.25).

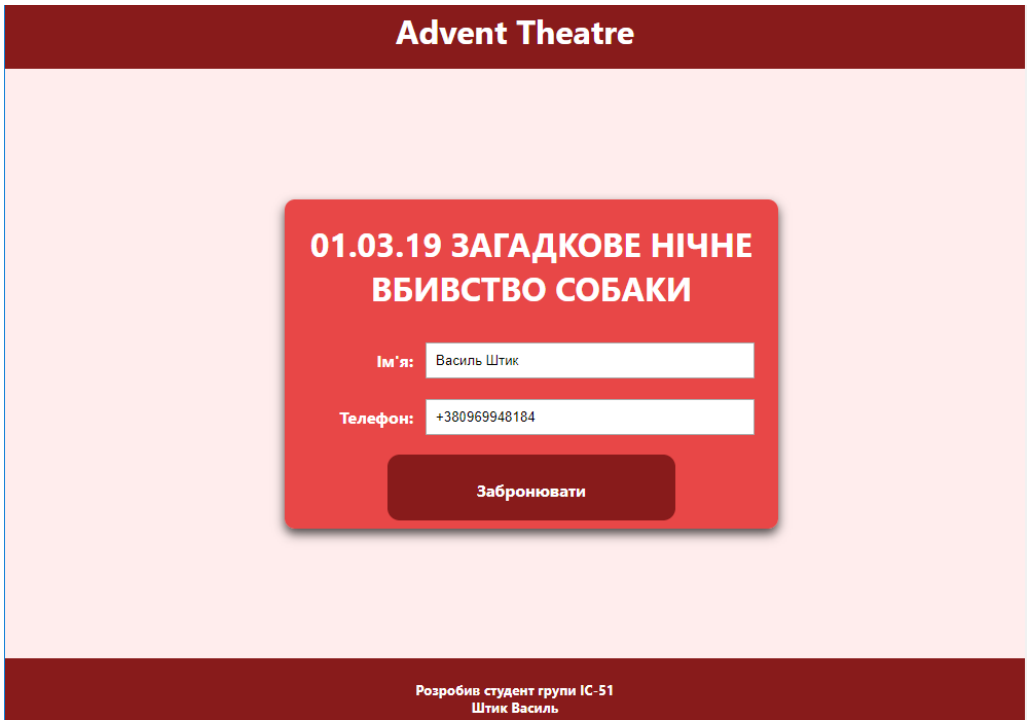


Рисунок 5.25 – Форма бронювання місця на заході

Після натискання на Забронювати, введені дані відправляються на API, результатом роботи якого буде створений в CRM запис Контакту та СМС повідомлення, надіслане за вказаним номером (рисунок 5.26).

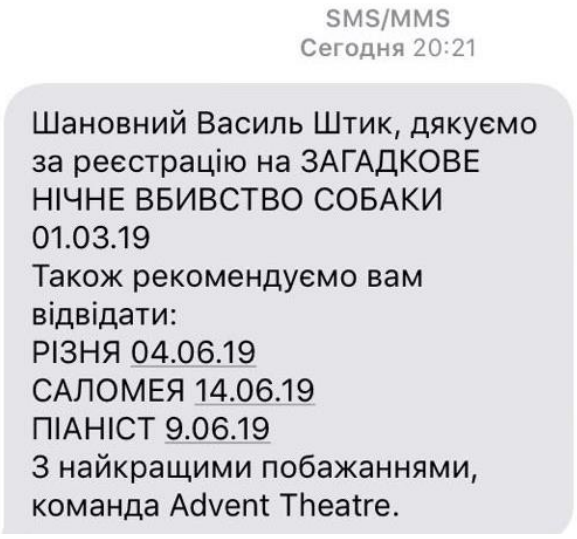


Рисунок 5.26 – СМС-повідомлення про успішне бронювання та списком рекомендованих заходів

5.2 Випробування програмного продукту

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій комплексу задач бронювання місць на театральні заходи та підбору рекомендованих заходів вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

В процесі тестування були перевірена уся функціональність системи. У наступних таблицях наведений перелік випробувань основних функціональних можливостей сайту (табл. 5.1 – 5.3).

Таблиця 5.1 – Результати тестування бронювання квитка

Назва:	Тест бронювання квитка	
Функція/ UseCase:	Бронювання квитка	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Відкрийте сайт	Сайт відкритий та доступний	Співпадає з очікуваним результатом
Кроки тесту:		
Натисніть на виставу на сайті	Відкривається вікно бронювання квитка на виставу	Співпадає з очікуваним результатом

Продовження Таблиці 5.1

Заповніть форму бронювання: “ Ім’я ”: Штик Василь, “Номер телефону”: +380969948184.	Дані успішно введені	Співпадає з очікуваним результатом
Натисніть кнопку “Забронювати»	На телефон приходить повідомлення про успішне бронювання	Співпадає з очікуваним результатом

Таблиця 5.2 – Результати тестування на коректність вводу даних

Назва:	Тест коректності вводу даних	
Функція/ UseCase:	Коректність вводу даних	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Відкрийте сайт	Сайт відкритий та доступний	Співпадає з очікуваним результатом
Кроки тесту:		
Натисніть на виставу на сайті	Відкривається вікно бронювання квитка на виставу	Співпадає з очікуваним результатом
Заповніть форму бронювання: “ Ім’я ”: SD%^vf; “Номер телефону”: +309948184.	Помилка вводу	Співпадає з очікуваним результатом

Таблиця 5.3 – Результати тестування натиснення на виставу

Назва:	Тест коректності вводу даних	
Функція/ UseCase:	Коректність вводу даних	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Відкрийте сайт	Сайт відкритий та доступний	Співпадає з очікуваним результатом

Продовження Таблиці 5.3

Кроки тесту:		
Натисніть на виставу на сайті	Відкривається вікно бронювання квитка на виставу	Співпадає з очікуваним результатом

У наступних таблицях наведений перелік випробувань основних функціональних можливостей CRM-системи (табл. 5.4 – 5.19).

Таблиця 5.4 – Результати тестування додання вистави в системі

Назва:	Тест додання вистави в CRM-мережу	
Функція/ UseCase:	Додання вистави в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Вистави».	Відкривається вікно вистав	Співпадає з очікуваним результатом
Натиснути на «Додати виставу»	Додання вистав	Співпадає з очікуваним результатом
Ввести інформацію про виставу	Ввести «Назва вистави», «Дата проведення», «Жанр», «Сцена», «Актори» та «Місце дії»	Співпадає з очікуваним результатом

Таблиця 5.5 – Результати тестування редагування вистави в системі

Назва:	Тест редагуванняння вистави в CRM-мережу	
Функція/ UseCase:	Редагування вистави в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом

Продовження Таблиці 5.5

Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Вистави».	Відкривається вікно вистав	Співпадає з очікуваним результатом
Натиснути на виставу та вибрати «Редагування»	Редагування вистав	Співпадає з очікуваним результатом
Змінити інформацію про виставу	Редагувати одне з полів «Назва вистави», «Дата проведення», «Жанр» та «Місце дії», «Сцена», «Актори»	Співпадає з очікуваним результатом

Таблиця 5.6 – Результати тестування видалення вистави в системі

Назва:	Тест видалення вистави в CRM-мережу	
Функція/ UseCase:	Видалення вистави в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Вистави».	Відкривається вікно вистав	Співпадає з очікуваним результатом
Натиснути на виставу та вибрати «Видалення»	Видалення вистави	Співпадає з очікуваним результатом
Підтвердження видалення	Вистава видалена	Співпадає з очікуваним результатом

Таблиця 5.7 – Результати тестування видалення заходів в системі

Назва:	Тест додавання заходів в CRM-мережу	
Функція/ UseCase:	Додавання заходів в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:

Продовження Таблиці 5.7

Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Заходи».	Відкривається вікно заходів	Співпадає з очікуваним результатом
Натиснути на заходи та вибрати «Видалити»	Видалення заходів	Співпадає з очікуваним результатом
Додати захід	Вибрати тип заходу: «Репетиції» та «Вистави»	Співпадає з очікуваним результатом

Таблиця 5.8 – Результати тестування редагування заходів в системі

Назва:	Тест редагування заходів в CRM-мережу	
Функція/ UseCase:	Редагування заходів в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Заходи».	Відкривається вікно заходів	Співпадає з очікуваним результатом
Натиснути на заходи та вибрати «Редагування»	Редагування заходів	Співпадає з очікуваним результатом
Редагувати захід	Додати поля «Назва вистави», «Дата проведення», «Жанр» та «Місце дії», «Сцена», «Актори»	Співпадає з очікуваним результатом

Таблиця 5.9 – Результати тестування видалення заходів в системі

Назва:	Тест видалення заходів в CRM-мережу	
Функція/ UseCase:	Видалення заходів в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумови:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки:		
Зайти в меню «Театр», вибрати з випадаючого списку «Заходи».	Відкривається вікно заходів	Співпадає з очікуваним результатом
Натиснути на заходи та вибрати «Видалити»	Видалення заходів	Співпадає з очікуваним результатом
Підтвердження видалення	Захід успішно видалено	Співпадає з очікуваним результатом

Таблиця 5.10 – Результати тестування додавання репетицій в системі

Назва:	Тест додавання репетицій в CRM-мережу	
Функція/ UseCase:	Додавання репетицій в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Репетиції».	Відкривається вікно репетицій	Співпадає з очікуваним результатом
Натиснути на репетиції та вибрати «Додавання»	Додавання репетицій	Співпадає з очікуваним результатом
Кроки тесту		
Додати репетицію	Ввести «День репетиції», «Час репетиції», «Назва вистави» та «Тривалість».	Співпадає з очікуваним результатом

Таблиця 5.11 – Результати тестування редагування репетицій в системі

Назва:	Тест редагування репетицій в CRM-мережу	
Функція/ UseCase:	Редагування репетицій в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Репетиції».	Відкривається вікно репетицій	Співпадає з очікуваним результатом
Натиснути на репетиції та вибрати «Редагування»	Редагування репетицій	Співпадає з очікуваним результатом
Редагувати репетицію	Змінити одне з полів «День репетиції», «Час репетиції», «Назва вистави» та «Тривалість».	Співпадає з очікуваним результатом

Таблиця 5.12 – Результати тестування видалення репетицій в системі

Назва:	Тест видалення репетицій в CRM-мережу	
Функція/ UseCase:	Видалення репетицій в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Репетиції».	Відкривається вікно репетицій	Співпадає з очікуваним результатом
Натиснути на репетиції та вибрати «Видалення»	Додавання репетицій	Співпадає з очікуваним результатом

Продовження Таблиці 5.12

Кроки тесту:		
Видалення репетицію	Репетицію успішно видалено	Співпадає з очікуваним результатом

Таблиця 5.13 – Результати тестування додавання репетицій в системі

Назва:	Тест додавання репетицій в CRM-мережу	
Функція/ UseCase:	Додавання репетицій в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Репетиції».	Відкривається вікно репетицій	Співпадає з очікуваним результатом
Натиснути на репетиції та вибрати «Додавання»	Додавання репетицій	Співпадає з очікуваним результатом
Додати репетицію	Ввести «День репетиції», «Час репетиції», «Назва вистави» та «Тривалість».	Співпадає з очікуваним результатом

Таблиця 5.14 – Результати тестування додавання репетицій в системі

Назва:	Тест додавання репетицій в CRM-мережу	
Функція/ UseCase:	Додавання репетицій в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом

Продовження Таблиці 5.14

Кроки тесту:		
Зайти у вкладку «Маркетинг», вибрати з випадаючого списку «Репетиції».	Відкривається вікно репетицій	Співпадає з очікуваним результатом
Натиснути на репетиції та вибрати «Додавання»	Додавання репетицій	Співпадає з очікуваним результатом
Додати репетицію	Ввести «День репетиції», «Час репетиції», «Назва вистави» та «Тривалість».	Співпадає з очікуваним результатом

Таблиця 5.15 – Результати тестування додавання ролі

Назва:	Тест додавання ролі	
Функція/ UseCase:	Додавання ролі в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Ролі».	Відкривається вікно ролей	Співпадає з очікуваним результатом
Натиснути на роль та вибрати «Додавання»	Додавання ролі	Співпадає з очікуваним результатом
Додати роль	Ввести «Ім’я» (Тип ролі)	Співпадає з очікуваним результатом

Таблиця 5.15 – Результати тестування видалення ролі

Назва:	Тест видалення ролі	
Функція/ UseCase:	Видалення ролі з CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:

Продовженяя Таблиці 5.15

Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Ролі».	Відкривається вікно ролей	Співпадає з очікуваним результатом
Натиснути на роль та вибрати «Видалення»	Видалення	Співпадає з очікуваним результатом
Видалити роль	Роль успішно виделено	Співпадає з очікуваним результатом

Таблиця 5.16 – Результати тестування додавання сцени

Назва:	Тест додавання сцени	
Функція/ UseCase:	Додавання сцени в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Сцени».	Відкривається вікно сцен	Співпадає з очікуваним результатом
Натиснути на сцену та вибрати «Додавання»	Додавання сцени	Співпадає з очікуваним результатом
Додати сцени	Ввести «Ім’я» (Тип сцени) та «Кількість місць»	Співпадає з очікуваним результатом

Таблиця 5.17 – Результати тестування видалення сцени

Назва:	Тест видалення сцени	
Функція/ UseCase:	Додавання сцени в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:

Продовження Таблиці 5.17

Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Сцени».	Відкривається вікно сцен	Співпадає з очікуваним результатом
Натиснути на сцену та вибрати «Видалити»	Видалити сцену	Співпадає з очікуваним результатом
Видалити сцени	Успішне видалення сцени	Співпадає з очікуваним результатом

Таблиця 5.18 – Результати тестування додавання жанра

Назва:	Тест додавання жанра	
Функція/ UseCase:	Додавання сжанра в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:
Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Жанри».	Відкривається вікно жанрів	Співпадає з очікуваним результатом
Натиснути на жанр та вибрати «Додавання»	Додавання жанру	Співпадає з очікуваним результатом
Додати жанр	Ввести «Ім’я»	Співпадає з очікуваним результатом

Таблиця 5.19 – Результати тестування видалити жанр

Назва:	Тест видалення жанру	
Функція/ UseCase:	Видалення жанру в CRM-мережу	
Дія:	Очікуваний результат:	Результат тесту:

Продовження Таблицю 5.19

Передумова:		
Залогінитись в CRM-системі	Вести e-mail та пароль від CRM-системи	Співпадає з очікуваним результатом
Кроки тесту:		
Зайти в меню «Театр», вибрати з випадаючого списку «Жанр».	Відкривається вікно жанрів	Співпадає з очікуваним результатом
Натиснути на жанр та вибрати «Видалення»	Видалення жанру	Співпадає з очікуваним результатом
Видалення жанру	Жанр успішно видалено	Співпадає з очікуваним результатом

Висновок до розділу

Цей розділ містить посібник користувача до програмного забезпечення, що було розроблено, як частина реалізації цього дипломного проекту. Система була перевірена за кількома критеріями. Перевірені системні функції, що виконуються системою, а саме додавання даних до CRM системи, перегляд запланованих заходів та репетицій працівником театру, перегляд афіши та бронювання місць на захід відвідувачем.

Окрім того, описано методи тестування створеного програмного забезпечення. Описані випробування та результати випробувань відповідають нормативним вимогам, що представлені в розділі «Загальні положення» цього дипломного проекту. Програмне забезпечення успішно пройшло випробування.

ЗАГАЛЬНІ ВИСНОВКИ

У ході виконання дипломного проекту було представлено опис функціональної моделі, де зазначені актори, та дії, які вони виконують в комплексі задач. На основі визначених варіантів використання, виявлені функціональні вимоги.

Проведено пошук та аналіз існуючих аналогів платформи, сформульовані призначення, цілі та задачі розробки.

Описано вхідні дані, що надходять від відвідувачів та адміністратора. Основні вихідні дані виводяться в CRM-систему, а для максимально простого інформування глядача було реалізовано надсилання СМС повідомлення. Також була представлена структура таблиць збереження даних в CRM-системі.

Були розглянуті використані інструменти розробки для реалізації програмного забезпечення системи підбору театральних заходів.

Було обгрунтовано вибір засобів розробки .Net, Phyton і Node.JS

Архітектура програмного забезпечення була представлена діаграмами класів, послідовностей і компонентів. Надана специфікація функцій, що реалізують процес підбору рекомендованих театральних заходів.

Визначено змістовне і математичне формулювання завдання підбору театральних подій на основі переваг усередненого відвідувача поточного заходу. Описано та обгрунтовано вибір методу вирішення поставленого завдання. Детально описано алгоритм підбору рекомендованих театральних подій. Надано докладний опис алгоритму, що використовуються для вирішення поставленої задачі.

Надано посібник користувача до програмного забезпечення. Перевірені системні функції, що виконуються системою. Окрім того, описані випробування та результати випробувань відповідають нормативним вимогам, що представлені в розділі «Загальні положення» цього дипломного проекту.

Програмне забезпечення успішно пройшло випробування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Театр — вид сценічного мистецтва // [Електронний ресурс] Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5% D0%B0%D1%82%D1%80
2. Що таке CRM-системи? // [Електронний ресурс] Режим доступу до ресурсу: https://habr.com/ru/company/trinion/blog/249633/
3. Як працюють рекомендаційні системи // [Електронний ресурс] Режим доступу до ресурсу: https://habr.com/ru/company/yandex/blog/241455/
4. Карабас — квиткове агентство. // [Електронний ресурс] Режим доступу до ресурсу: https://karabas.com/
5. Concert.ua — квитковий сервіс. // [Електронний ресурс] Режим доступу до ресурсу: https://concert.ua/
6. Афіша Яндекс — розклад всіх заходів міста. // [Електронний ресурс] Режим доступу до ресурсу: https://afisha.yandex.ua/
7. KudaGo — сервіс по просуванню подій. // [Електронний ресурс] Режим доступу до ресурсу: https://kudago.com/
8. Інтелектуальні бізнес-додатки. // [Електронний ресурс] Режим доступу до ресурсу: https://dynamics.microsoft.com/en-us/
9. Что такое API. // [Електронний ресурс] Режим доступу до ресурсу: https://habr.com/ru/sandbox/52599/
10. ASP.NET Core 1.1 Web API For Beginners: How To Build a Web API/ Jon Galloway, Benjamin Perkins, Shayne Boyer, Filip Wojcieszyn, K. Scott Allen, Wrox, 2019, 624 с.
11. Flask is a microframework for Python. // [Електронний ресурс] Режим доступу до ресурсу: http://flask.pocoo.org/
12. Блог Миндубаева Рамазана. // [Електронний ресурс] Режим доступу до ресурсу: http://mindubaev.com/100-dnej/chto-takoe-tf-idf/

Додаток А

Тексти програмного коду

Автоматизована рекомендаційна система підбору театральних подій

(Найменування програми (документа))

DVD-R

(Вид носія даних)

(Обсяг програми (документа) , арк.,) Мб)

Київ – 2019 року

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AdventTheatre.Models;
using AdventTheatre.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Microsoft.Xrm.Sdk;

namespace AdventTheatre.Controllers
{
    [Produces("application/json")]
    [Route("api/[controller]")]
    public class SpectacleController : BaseCrmController
    {
        public SpectacleController(IOrganizationService crmService,
ILogger<SpectacleController> logger,
            ISpectacleService spectacleService) : base(crmService, logger,
spectacleService)
        {
        }

        [HttpGet]
        public IActionResult Get()
        {
            _logger.LogWarning("Service is available");
            return Ok("Service is available");
        }

        [HttpPost("book")]
        public IActionResult Book([FromBody] BookingModel bookingDetail)
        {
            if (bookingDetail is null) _logger.LogError("Not valid model");

            try
            {
                _spectacleService.Book(bookingDetail, _crmService);
                _logger.LogWarning("Book is executed");
                return Ok("Executed");
            }
            catch (Exception e)
            {
                _logger.LogError(e, e.Message);
                return StatusCode(500, e);
            }
        }
    }
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AdventTheatre.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Microsoft.Xrm.Sdk;

namespace AdventTheatre.Controllers
{
    public class BaseCrmController : Controller
    {
        protected readonly IOrganizationService _crmService;
        protected readonly ILogger<SpectacleController> _logger;
```

```
        protected readonly ISpectacleService _spectacleService;

        protected BaseCrmController(
            IOrganizationService crmService,
            ILogger<SpectacleController> logger,
            ISpectacleService spectacleService)
        {
            _spectacleService = spectacleService;
            _logger = logger;
            _crmService = crmService;
        }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace AdventTheatre.Models
{
    public class BookingModel
    {
        private string phone;

        public string Name { get; set; }

        public string EventId { get; set; }

        public string Phone
        {
            get => phone;
            set
            {
                string result = Regex.Replace(value, @"[^\d]", "");

                // MobilePhone like 098...
                Regex regEx = new Regex("^[0][0-9]{9}$");

                if (regEx.IsMatch(result))
                {
                    phone = "+38" + result;
                    return;
                }

                // MobilePhone like 38098...
                regEx = new Regex("^[3][8][0][0-9]{9}$");

                if (regEx.IsMatch(result))
                {
                    phone = "+" + result;
                    return;
                }

                // MobilePhone like 98...
                regEx = new Regex("^[0-9]{9}$");

                if (regEx.IsMatch(result))
                {
                    phone = "+380" + result;
                    return;
                }
```

```
                // MobilePhone like 8098...
                regEx = new Regex("^[8][0][0-9]{9}$");

                if (regEx.IsMatch(result))
                {
                    phone = "+3" + result;
                    return;
                }

                throw new ValidationException("Incorrect Phone Number!");
            }

        }
    }
}
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;

namespace AdventTheatre.Models
{
        // Do not modify the content of this file.
        // This is an automatically generated file and all
        // logic should be added in the assocaited controller class
        // If a controller does not exist, create one that inherrits the model.

        public class ContactModel : EntityBase
        {
                // Public static Logical Name
                public const string
                        LogicalName = "contact";

                #region Attribute Names
                public static class Fields
                {
                        public const string
                                Address1Phone = "address1_telephone1",
                                Description = "description",
                                FirstName = "firstname",
                                FullName = "fullname",
                                MobilePhone = "mobilephone",
                                PrimaryId = "contactid";

                        public static string[] All => new[] { Address1Phone,
                                Description,
                                FirstName,
                                FullName,
                                MobilePhone,
                                PrimaryId };
                }
                #endregion

                #region Enums

                #endregion

                #region Field Definitions
                public string Address1Phone
                {
                        get { return (string)this[Fields.Address1Phone]; }
                        set { this[Fields.Address1Phone] = value; }
                }
```

```
                public string Description
                {
                        get { return (string)this[Fields.Description]; }
                        set { this[Fields.Description] = value; }
                }
                public string FirstName
                {
                        get { return (string)this[Fields.FirstName]; }
                        set { this[Fields.FirstName] = value; }
                }
                public string FullName
                {
                        get { return (string)this[Fields.FullName]; }
                        set { this[Fields.FullName] = value; }
                }
                public string MobilePhone
                {
                        get { return (string)this[Fields.MobilePhone]; }
                        set { this[Fields.MobilePhone] = value; }
                }
                #endregion

                #region Constructors
                protected ContactModel()
                        : base(LogicalName) { }
                protected ContactModel(IOrganizationService service)
                        : base(LogicalName, service) { }
                protected ContactModel(Guid id, ColumnSet columnSet, IOrganizationService
service)
                        : base(service.Retrieve(LogicalName, id, columnSet), service) { }
                protected ContactModel(Guid id, IOrganizationService service)
                        : base(LogicalName, id, service) { }
                protected ContactModel(Entity entity, IOrganizationService service)
                        : base(entity, service) { }
                #endregion
        }
}
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Messages;
using Microsoft.Xrm.Sdk.Query;

namespace AdventTheatre.Models
{
    public class EntityBase
    {
        private IDictionary<string, object> _changes = new Dictionary<string, object>();
        private Guid? _entityGuid;
        private readonly string _logicalName;
        // ReSharper disable once InconsistentNaming
        protected readonly IOrganizationService _service;
        // Private Attributes
        private readonly IDictionary<string, object> _values = new Dictionary<string,
object>();

        // Public Attributes
        private bool _preOperation;
        private Entity _targetEntity;

        // Constructors
```

```
/// <summary>
///     Constructor used to setup an exisitng entity
/// </summary>
/// <param name="record"></param>
/// <param name="service"></param>
protected EntityBase(Entity record, IOrganizationService service)
{
    // Process ID
    _entityGuid = record.Id;
    _logicalName = record.LogicalName;
    _service = service;

    // Process attributes
    foreach (var attribute in record.Attributes)
        _values.Add(attribute);
}

/// <summary>
///     Constructor for a new entity.
/// </summary>
/// <param name="logicalName"></param>
/// <param name="service"></param>
protected EntityBase(string logicalName, IOrganizationService service = null)
{
    _logicalName = logicalName;
    _service = service;
}

/// <summary>
///     Constructor for existing entity - however, get no attributes!
/// </summary>
/// <param name="logicalName"></param>
/// <param name="id"></param>
/// <param name="service"></param>
protected EntityBase(string logicalName, Guid id, IOrganizationService service)
{
    _logicalName = logicalName;
    _entityGuid = id;
    _service = service;
}

// Public Atttributes
public Guid? Id
{
    get { return _entityGuid; }
    set { _entityGuid = value; }
}

// Accessor for attributes
public object this[string attributeName]
{
    get
    {
        return _values.ContainsKey(attributeName)
            ? _values[attributeName]
            : null;
    }
    set
    {
        // Set the current value
        if (_values.ContainsKey(attributeName))
            _values[attributeName] = value;
        else
```

```
                    _values.Add(attributeName, value);

                // Set the changeset
                if (_changes.ContainsKey(attributeName))
                    _changes[attributeName] = value;
                else
                    _changes.Add(attributeName, value);
            }
        }

        // Fields
        public OptionSetValue Status
        {
            get { return (OptionSetValue)this[CoreFields.StatusField]; }
            set { this[CoreFields.StatusField] = value; }
        }

        public OptionSetValue StatusReason
        {
            get { return (OptionSetValue)this[CoreFields.StatusReasonField]; }
            set { this[CoreFields.StatusReasonField] = value; }
        }

        // Public methods

        public bool IsDirty()
        {
            return _changes.Count > 0;
        }

        /// <summary>
        /// Register the entity object for preoperation mode
        /// This allows plugins to use "Save()" where required
        /// and only update the target entity reference
        /// </summary>
        /// <param name="target">Target entity of the plugin</param>
        public void RegisterAsPreOperation(Entity target)
        {
            _targetEntity = target;
            _preOperation = true;
        }

        /// <summary>
        ///     Save the record - be this a Create or Update, only changes sent
        ///     will be pushed to the server.
        /// </summary>
        public void Save()
        {
            // Validate service object
            if (_service == null)
                throw new Exception("Unable to save an entity object without a service
object");

            // 21.10.2015, RA:  Updated to allow for preoperation plugins
            //                  to correctly update the target entity
            if (_preOperation && _targetEntity != null)
            {
                foreach (var c in _changes)
                    _targetEntity[c.Key] = c.Value;
                return;
            }

            // Create the new Entity object
```

```
            var obj = new Entity(_logicalName);

            // Add all changes attributes;
            obj.Attributes.AddRange(_changes);

            // Check if this is an Update
            if (_entityGuid != Guid.Empty && _entityGuid != null)
            {
                // RA:  27.11.2016  -   Ensure there are changes!
                if (_changes.Count == 0)
                    return;
                // END

                obj.Id = (Guid)_entityGuid;
                _service.Update(obj);
            }
            else
            {
                // This is an Create
                _entityGuid = _service.Create(obj);
            }
            _changes = new Dictionary<string, object>();
        }

        /// <summary>
        ///     Function to delete the current entity
        /// </summary>
        public void Delete()
        {
            // Check the id
            if (_entityGuid != null)
                _service.Delete(_logicalName, (Guid)_entityGuid);
        }

        /// <summary>
        ///     Get multiple attributes on the current entity.
        /// </summary>
        /// <param name="attributes">String array of attributes</param>
        public EntityBase Get(string[] attributes)
        {
            // If we have no id, do nothing
            if (_entityGuid == null)
                return null;

            // Get the attributes;
            Entity entity;
            try
            {
                entity = _service.Retrieve(
                    _logicalName,
                    (Guid)_entityGuid,
                    new ColumnSet(attributes)
                );
            }
            catch
            {
                return null;
            }

            // Foreach attribute, add to the entity;
            foreach (var attribute in attributes)
                if (_values.ContainsKey(attribute))
                    _values[attribute] =
```

```
                        entity.Contains(attribute)
                            ? entity[attribute]
                            : null;
                else
                    _values.Add(
                        attribute,
                        entity.Contains(attribute)
                            ? entity[attribute]
                            : null
                    );

            // We have updated the entity..
            // Remove all changes relating to the fields updated
            var tmp = _changes;
            _changes = new Dictionary<string, object>();
            foreach (var change in tmp.Where(change => !attributes.Contains(change.Key)))
                _changes.Add(change);

            // Finally return this object
            return this;
        }

        /// <summary>
        ///     Get a single attribute at the current entity
        /// </summary>
        /// <param name="attribute">string of attribute to retreive</param>
        public object Get(string attribute)
        {
            Get(new[] { attribute });
            return this[attribute];
        }

        /// <summary>
        ///     Gets a record based on parameters sent
        /// </summary>
        /// <param name="id">GUID of the record</param>
        /// <param name="columnSet">Column Set - Which fields are needed?</param>
        /// <param name="logicalName">Logical Name of Entity to Retreive</param>
        /// <param name="service">Service object</param>
        /// <returns></returns>
        public static EntityBase GetRecord(Guid id, ColumnSet columnSet, string
logicalName,
            IOrganizationService service)
        {
            var tmpEntity = service.Retrieve(
                logicalName,
                id,
                columnSet
            );
            return new EntityBase(tmpEntity, service);
        }

        /// <summary>
        ///     Return a copy of the record
        /// </summary>
        /// <param name="entity"></param>
        /// <returns></returns>
        public Entity CopyRecord(Entity entity)
        {
            var obj = entity;
            obj.Id = Guid.Empty;
            obj.Attributes.Remove(entity.LogicalName + "id");
```

```
            //this._service.Create(obj);

            return obj;
        }

        public Entity CopyRecord(ITracingService tracer = null)
        {
            var obj = new Entity(_logicalName) { Id = Guid.Empty };
            obj.Attributes.AddRange(_values.ToArray());
            tracer?.Trace("COPY RECORD: No. attributes: " + obj.Attributes.Count);
            obj.Attributes.Remove(_logicalName + "id");
            return obj;
        }

        /// <summary>
        ///     Returns a delete request for the current object
        /// </summary>
        /// <returns></returns>
        public DeleteRequest DeleteRequest()
        {
            if (_entityGuid == null)
                return null;

            // return the delete request
            return new DeleteRequest
            {
                Target = GetReference()
            };
        }

        /// <summary>
        ///     Returns an update request for the current object
        /// </summary>
        /// <returns></returns>
        public UpdateRequest UpdateRequest()
        {
            if (_entityGuid == null)
                return null;

            // RA:  27.11.2016  -   Again, ensure there are changes!
            if (_changes.Count == 0)
                return null;
            // END

            // Return the update, this can then be bulk updated
            return new UpdateRequest { Target = GetEntity() };
        }

        /// <summary>
        ///     Returns a create request for the current object
        /// </summary>
        /// <returns></returns>
        public CreateRequest CreateRequest()
        {
            // Return the update, this can then be bulk updated
            return new CreateRequest { Target = GetEntity() };
        }

        /// <summary>
        ///     Bulk Create Functionality
        /// </summary>
        /// <param name="request"></param>
        /// <param name="ts"></param>
```

```
        protected void BulkCreate(ExecuteMultipleRequest request, ITracingService ts =
null)
        {
            var responseWithNoResults =
(ExecuteMultipleResponse)_service.Execute(request);
            if (ts != null)
            {
                ts.Trace("Display errors");
                foreach (var responseItem in responseWithNoResults.Responses)
                    if (responseItem.Fault != null)
                        ts.Trace("A fault occurred when processing " +
                                 request.Requests[responseItem.RequestIndex].RequestName
+
                                 " request, with a fault message: " +
responseItem.Fault.Message);
            }
            request.Requests.Clear();
        }

        public Entity GetEntity()
        {
            // Create the new Entity object
            var obj = new Entity(_logicalName);

            // Add all changes attributes;
            obj.Attributes.AddRange(_changes);

            // Add the ID
            if (_entityGuid != null)
                obj.Id = (Guid)_entityGuid;

            return obj;
        }

        public Entity GetFullEntity()
        {
            // Create the new Entity object
            var obj = new Entity(_logicalName);

            // Add all changes attributes;
            obj.Attributes.AddRange(_values);

            // Add the ID
            if (_entityGuid != null)
                obj.Id = (Guid)_entityGuid;

            return obj;
        }

        public EntityReference GetReference()
        {
            if (Id == null)
                throw new Exception("Unable to get Reference of a non created entity");
            return new EntityReference(_logicalName, Id.Value);
        }

        public string GetLogicalName()
        {
            if (_logicalName == null)
                throw new ArgumentNullException(nameof(_logicalName));
            return _logicalName;
        }
```

```
        public static class CoreFields
        {
            public static string
                StatusField = "statecode",
                StatusReasonField = "statuscode",
                CreatedByField = "createdby",
                CreatedOnField = "createdon",
                ModifiedByField = "modifiedby",
                ModifiedOnField = "modifiedon",
                OwnerIdField = "ownerid";
        }
    }
}
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;

namespace AdventTheatre.Models
{
	// Do not modify the content of this file.
	// This is an automatically generated file and all
	// logic should be added in the assocaited controller class
	// If a controller does not exist, create one that inherrits the model.

	public class EventModel : EntityBase
	{
		// Public static Logical Name
		public const string
			LogicalName = "at_event";

		#region Attribute Names
		public static class Fields
		{
			public const string
				PrimaryId = "at_eventid",
				Spectacle = "at_spectacleid",
				Date = "at_datetime",
				Name = "at_name",
				Scene = "at_sceneid";

			public static string[] All => new[] { PrimaryId,
				Spectacle,
				Date,
				Name,
				Scene };
		}
		#endregion

		#region Enums

		#endregion

		#region Field Definitions
		public EntityReference Spectacle
		{
			get { return (EntityReference)this[Fields.Spectacle]; }
			set { this[Fields.Spectacle] = value; }
		}
		public DateTime? Date
		{
			get { return (DateTime?)this[Fields.Date]; }
			set { this[Fields.Date] = value; }
		}
```

```
                public string Name
                {
                        get { return (string)this[Fields.Name]; }
                        set { this[Fields.Name] = value; }
                }
                public EntityReference Scene
                {
                        get { return (EntityReference)this[Fields.Scene]; }
                        set { this[Fields.Scene] = value; }
                }
                #endregion

                #region Constructors
                protected EventModel()
                        : base(LogicalName) { }
                protected EventModel(IOrganizationService service)
                        : base(LogicalName, service) { }
                protected EventModel(Guid id, ColumnSet columnSet, IOrganizationService
service)
                        : base(service.Retrieve(LogicalName, id, columnSet), service) { }
                protected EventModel(Guid id, IOrganizationService service)
                        : base(LogicalName, id, service) { }
                protected EventModel(Entity entity, IOrganizationService service)
                        : base(entity, service) { }
                #endregion
        }
}
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;

namespace AdventTheatre.Models
{
        // Do not modify the content of this file.
        // This is an automatically generated file and all
        // logic should be added in the assocaited controller class
        // If a controller does not exist, create one that inherrits the model.

        public class GenreModel : EntityBase
        {
                // Public static Logical Name
                public const string
                        LogicalName = "at_genre";

                #region Attribute Names
                public static class Fields
                {
                        public const string
                                PrimaryId = "at_genreid",
                                Name = "at_name";

                        public static string[] All => new[] { PrimaryId,
                                Name };
                }
                #endregion

                #region Enums

                #endregion

                #region Field Definitions
                public string Name
                {
```

```
                        get { return (string)this[Fields.Name]; }
                        set { this[Fields.Name] = value; }
                }
                #endregion

                #region Constructors
                protected GenreModel()
                        : base(LogicalName) { }
                protected GenreModel(IOrganizationService service)
                        : base(LogicalName, service) { }
                protected GenreModel(Guid id, ColumnSet columnSet, IOrganizationService
service)
                        : base(service.Retrieve(LogicalName, id, columnSet), service) { }
                protected GenreModel(Guid id, IOrganizationService service)
                        : base(LogicalName, id, service) { }
                protected GenreModel(Entity entity, IOrganizationService service)
                        : base(entity, service) { }
                #endregion
        }
}
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;

namespace AdventTheatre.Models
{
        // Do not modify the content of this file.
        // This is an automatically generated file and all
        // logic should be added in the assocaited controller class
        // If a controller does not exist, create one that inherrits the model.

        public class SpectacleModel : EntityBase
        {
                // Public static Logical Name
                public const string
                        LogicalName = "at_spectacle";

                #region Attribute Names
                public static class Fields
                {
                        public const string
                                PrimaryId = "at_spectacleid",
                                CreatedBy = "createdby",
                                Autor = "at_autor",
                                Name = "at_name",
                                Director = "at_director";

                        public static string[] All => new[] { PrimaryId,
                  CreatedBy,
                                Autor,
                                Name,
                                Director };
                }
                #endregion

                #region Enums

                #endregion

                #region Field Definitions
                public EntityReference CreatedBy
         {
                        get { return (EntityReference)this[Fields.CreatedBy]; }
```

```
                set { this[Fields.CreatedBy] = value; }
            }
            public EntityReference Autor
            {
                get { return (EntityReference)this[Fields.Autor]; }
                set { this[Fields.Autor] = value; }
            }
            public string Name
            {
                get { return (string)this[Fields.Name]; }
                set { this[Fields.Name] = value; }
            }
            public EntityReference Director
            {
                get { return (EntityReference)this[Fields.Director]; }
                set { this[Fields.Director] = value; }
            }
            #endregion

            #region Constructors
            protected SpectacleModel()
                : base(LogicalName) { }
            protected SpectacleModel(IOrganizationService service)
                : base(LogicalName, service) { }
            protected SpectacleModel(Guid id, ColumnSet columnSet, IOrganizationService
service)
                : base(service.Retrieve(LogicalName, id, columnSet), service) { }
            protected SpectacleModel(Guid id, IOrganizationService service)
                : base(LogicalName, id, service) { }
            protected SpectacleModel(Entity entity, IOrganizationService service)
                : base(entity, service) { }
            #endregion
        }
} using System;
using System.Collections.Generic;
using System.Linq;
using AdventTheatre.Models;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;

namespace AdventTheatre.EntityProviders
{
    public class Contact : ContactModel
    {
        public Contact(IOrganizationService service) : base(service)
        {
        }

        public Contact(Guid id, IOrganizationService service) : base(id, service)
        {
        }

        public Contact(Entity entity, IOrganizationService service) : base(entity,
service)
        {
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AdventTheatre.Models;
```

```
using Microsoft.Xrm.Sdk;

namespace AdventTheatre.EntityProvider
{
    public class Event: EventModel
    {
        public Event(IOrganizationService service) : base(service)
        {
        }

        public Event(Guid id, IOrganizationService service) : base(id, service)
        {
        }

        public Event(Entity entity, IOrganizationService service) : base(entity, service)
        {
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AdventTheatre.Models;
using Microsoft.Xrm.Sdk;

namespace AdventTheatre.EntityProvider
{
    public class Genre: GenreModel
    {
        public Genre(IOrganizationService service) : base(service)
        {
        }

        public Genre(Guid id, IOrganizationService service) : base(id, service)
        {
        }

        public Genre(Entity entity, IOrganizationService service) : base(entity, service)
        {
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AdventTheatre.Models;
using Microsoft.Xrm.Sdk;

namespace AdventTheatre.EntityProvider
{
    public class Spectacle: SpectacleModel
    {
        public Spectacle(IOrganizationService service) : base(service)
        {
        }

        public Spectacle(Guid id, IOrganizationService service) : base(id, service)
        {
        }

        public Spectacle(Entity entity, IOrganizationService service) : base(entity,
service)
        {
        }
    }
}

using Microsoft.Xrm.Sdk;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AdventTheatre.Models;

namespace AdventTheatre.Services
{
    public interface ISpectacleService
    {
        void Book(BookingModel bookingdetail, IOrganizationService service);
    }
```

```
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Threading.Tasks;
using AdventTheatre.EntityProvider;
using AdventTheatre.EntityProviders;
using AdventTheatre.Models;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Messages;
using Microsoft.Xrm.Sdk.Query;

namespace AdventTheatre.Services
{
    public sealed class SpectacleService : ISpectacleService
    {
        public void Book(BookingModel bookingDetail, IOrganizationService service)
        {
            var query = new QueryExpression(ContactModel.LogicalName)
            {
                ColumnSet = new ColumnSet()
            };
            query.Criteria.AddCondition(ContactModel.Fields.MobilePhone,
ConditionOperator.Like, bookingDetail.Phone);
            var contacts=service.RetrieveMultiple(query);

            if (contacts.Entities.Count>0)
            {
                AssociateRequest contactToEvent = new AssociateRequest
                {
                    Target = new EntityReference(EventModel.LogicalName, new
Guid(bookingDetail.EventId)),
                    RelatedEntities = new EntityReferenceCollection
                    {
                        new EntityReference(ContactModel.LogicalName,
contacts.Entities.First().Id)
                    },
                    Relationship = new Relationship("at_at_event_contact")
                };

                // Execute the request.
                service.Execute(contactToEvent);
                //TODO првязку к мероприятию
            }
            else
            {
                var currentEvent = new Event(service.Retrieve(EventModel.LogicalName, new
Guid(bookingDetail.EventId),
                    new ColumnSet(EventModel.Fields.All)),service);

                var recomendedList = GetRecomendation(currentEvent, service);

                var contact = new Contact(service)
                {
                    FirstName = bookingDetail.Name,
                    MobilePhone = bookingDetail.Phone,
                    Description=@"Шановний "+bookingDetail.Name+" дякуємо за реєстрацію
на "+currentEvent.Name+"."+
                                  "Також рекомендуємо вам відвідати:"+
                                  recomendedList[0]+
                                  recomendedList[1]+
                                  recomendedList[2]+"."
```

```
                };
                contact.Save();
                var id=contact.Id;

                AssociateRequest contactToEvent = new AssociateRequest
                {
                    Target = new EntityReference(EventModel.LogicalName, new
Guid(bookingDetail.EventId)),
                    RelatedEntities = new EntityReferenceCollection
                    {
                        new EntityReference(ContactModel.LogicalName, id.Value)
                    },
                    Relationship = new Relationship("at_at_event_contact")
                };

                // Execute the request.
                service.Execute(contactToEvent);
            }

        }

        private List<string> GetRecomendation(Event currentEvent, IOrganizationService
service)
        {
            var list= new List<string>();

            var currentSpectacleTags = GetSpectacleTags(currentEvent.Spectacle.Id,
service);

            var spectaclesTags = GetTags(currentEvent, service);

            //var rating = GetRating(currentSpectacleTags, spectaclesTags);
            var frequency = GetSpectacleFrequency(currentEvent, service);
            var orderedFrequncy=frequency.OrderBy(e => e.Value);

            foreach (var spectacleDict in orderedFrequncy.Take(3).ToList())
            {
                var spectacle = new Spectacle(service.Retrieve(Spectacle.LogicalName,
spectacleDict.Key,
                    new ColumnSet(SpectacleModel.Fields.Name)), service);

                list.Add(spectacle.Name);
            }
            return list;
        }

        private Dictionary<Guid, int> GetRating(List<string> currentSpectacleTags,
Dictionary<Guid, List<string>> spectaclesTags)
        {
            var ratingDictionary = new Dictionary<Guid, int>();

            return ratingDictionary;

        }

        private Dictionary<Guid, List<string>> GetTags(Event currentEvent,
IOrganizationService service)
        {
            var edDictionary=new Dictionary<Guid, List<string>>();

            var visitors = GetVisitors(currentEvent.Spectacle.Id, service);

            foreach (var visitor in visitors)
```

```
            {
                var visitorSpectacles = getVisitorSpectacles(visitor, service);

                foreach (var spectacle in visitorSpectacles)
                {
                    if (!edDictionary.ContainsKey(spectacle.Id.Value))
                    {
                        edDictionary.Add(spectacle.Id.Value,
GetSpectacleTags(spectacle.Id.Value, service));
                    }
                }
            }

            edDictionary.Remove(currentEvent.Spectacle.Id);
            return edDictionary;
        }

        private Dictionary<Guid, int> GetSpectacleFrequency(Event currentEvent,
IOrganizationService service)
        {
            var edDictionary = new Dictionary<Guid, int>();

            var visitors = GetVisitors(currentEvent.Spectacle.Id, service);

            foreach (var visitor in visitors)
            {
                var visitorSpectacles = getVisitorSpectacles(visitor, service);

                foreach (var spectacle in visitorSpectacles)
                {
                    if (!edDictionary.ContainsKey(spectacle.Id.Value))
                    {
                        edDictionary.Add(spectacle.Id.Value, 1);
                    }
                    else
                    {
                        var n = edDictionary[spectacle.Id.Value];
                        edDictionary.Add(spectacle.Id.Value, n++);
                    }
                }
            }

            edDictionary.Remove(currentEvent.Spectacle.Id);
            return edDictionary;
        }

        private List<Spectacle> getVisitorSpectacles(Contact visitor,
IOrganizationService service)
        {
            var list=new List<Spectacle>();
            var collRecords=new List<Event>();

            string relationshipEntityName = "at_at_event_contact";
            var query = new QueryExpression(EventModel.Fields.Spectacle);
            query.ColumnSet = new ColumnSet(true);
            var linkEntity1 = new LinkEntity(EventModel.LogicalName,
relationshipEntityName, "at_eventid", "at_eventid",  JoinOperator.Inner);
            var linkEntity2 = new LinkEntity(relationshipEntityName,
ContactModel.LogicalName, "at_contactid", "at_contactid", JoinOperator.Inner);
            linkEntity1.LinkEntities.Add(linkEntity2);
            query.LinkEntities.Add(linkEntity1);
            linkEntity2.LinkCriteria = new FilterExpression();
```

```
            linkEntity2.LinkCriteria.AddCondition(new ConditionExpression("contactid",
ConditionOperator.Equal, visitor.Id));
            collRecords.AddRange(service.RetrieveMultiple(query).Entities.Select(e => new
Event(e, service)).ToList());

            foreach (var eEvent in collRecords)
            {
                list.Add(new Spectacle(eEvent.Spectacle.Id, service));
            }
            return list;
        }

        private List<string> GetSpectacleTags(Guid id, IOrganizationService service)
        {
            var edDictionary = new Dictionary<Guid, List<string>>();
            var currentSpectacleTags = new List<string>();

            var spectacle = new Spectacle(service.Retrieve(SpectacleModel.LogicalName,
id,
                new ColumnSet(SpectacleModel.Fields.All)), service);

            currentSpectacleTags.Add(spectacle.Autor.Name);
            currentSpectacleTags.Add(spectacle.Director.Name);
            currentSpectacleTags.Add(GetGeneres(spectacle.Id.Value, service));
            currentSpectacleTags.Add(GetActors(spectacle.Id.Value, service));

            return currentSpectacleTags;
        }

        private string GetActors(Guid id, IOrganizationService service)
        {
            string relationshipEntityName = "at_at_spectacle_contact";
            var query = new QueryExpression(ContactModel.LogicalName);
            query.ColumnSet = new ColumnSet(true);
            var linkEntity1 = new LinkEntity(ContactModel.LogicalName,
relationshipEntityName, "contactid", "contactid", JoinOperator.Inner);
            var linkEntity2 = new LinkEntity(relationshipEntityName,
SpectacleModel.LogicalName, "at_spectacleid", "at_spectacleid", JoinOperator.Inner);
            linkEntity1.LinkEntities.Add(linkEntity2);
            query.LinkEntities.Add(linkEntity1);
            linkEntity2.LinkCriteria = new FilterExpression();
            linkEntity2.LinkCriteria.AddCondition(new
ConditionExpression("at_spectacleid", ConditionOperator.Equal, id));
            var collRecords = service.RetrieveMultiple(query).Entities.Select(e => new
Contact(e, service)).ToList();

            var actors = "";
            foreach (var tag in collRecords)
            {
                actors += tag.FullName + ";";
            }

            return actors;
        }

        private string GetGeneres(Guid id, IOrganizationService service)
        {
            string relationshipEntityName = "at_at_genre_at_spectacle";
            var query = new QueryExpression(GenreModel.LogicalName);
            query.ColumnSet = new ColumnSet(true);
            var linkEntity1 = new LinkEntity(GenreModel.LogicalName,
relationshipEntityName, "at_genreid", "at_genreid", JoinOperator.Inner);
```

```
            var linkEntity2 = new LinkEntity(relationshipEntityName,
SpectacleModel.LogicalName, "at_spectacleid", "at_spectacleid", JoinOperator.Inner);
            linkEntity1.LinkEntities.Add(linkEntity2);
            query.LinkEntities.Add(linkEntity1);
            linkEntity2.LinkCriteria = new FilterExpression();
            linkEntity2.LinkCriteria.AddCondition(new
ConditionExpression("at_spectacleid", ConditionOperator.Equal, id));
            var collRecords = service.RetrieveMultiple(query).Entities.Select(e => new
Genre(e, service)).ToList();

            var generes = "";
            foreach (var tag in collRecords)
            {
                generes += tag.Name + ";";
            }

            return generes;
        }

        private List<Contact> GetVisitors(Guid id, IOrganizationService service)
        {
            var collRecords= new List<Contact>();
            var query= new QueryExpression(EventModel.LogicalName);
            query.ColumnSet = new ColumnSet();
            query.Criteria.AddCondition(EventModel.Fields.Spectacle,
ConditionOperator.Equal, id);
            var events = service.RetrieveMultiple(query).Entities.Select(e=>new Event(e,
service)).ToList();
            foreach (var eEvent in events)
            {
                string relationshipEntityName = "at_at_event_contact";
                query = new QueryExpression(ContactModel.LogicalName);
                query.ColumnSet = new ColumnSet(true);
                var linkEntity1 = new LinkEntity(ContactModel.LogicalName,
relationshipEntityName, "at_contactid", "at_contactid", JoinOperator.Inner);
                var linkEntity2 = new LinkEntity(relationshipEntityName,
EventModel.LogicalName, "at_eventid", "at_eventid", JoinOperator.Inner);
                linkEntity1.LinkEntities.Add(linkEntity2);
                query.LinkEntities.Add(linkEntity1);
                linkEntity2.LinkCriteria = new FilterExpression();
                linkEntity2.LinkCriteria.AddCondition(new
ConditionExpression("at_eventid", ConditionOperator.Equal, id));
                collRecords.AddRange(service.RetrieveMultiple(query).Entities.Select(e =>
new Contact(e, service)).ToList());
            }


            return collRecords;
        }
    }
}
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace AdventTheatre
{
```

```
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateWebHostBuilder(args).Build().Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args).ConfigureLogging(loggingBuilder =>
            {
                loggingBuilder.AddAzureWebAppDiagnostics();
            }).UseStartup<Startup>();

    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AdventTheatre.Services;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Tooling.Connector;
using Swashbuckle.AspNetCore.Swagger;

namespace AdventTheatre
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the
container.
        public void ConfigureServices(IServiceCollection services)
        {
            var crmConfig = Configuration.GetConnectionString("CrmConnectionString");

            services.AddScoped<IOrganizationService, CrmServiceClient>(provider => new
CrmServiceClient(crmConfig));
            services.AddScoped<ISpectacleService, SpectacleService>();

            //services.AddMvc().AddMvcOptions(options =>
            //options.ModelMetadataDetailsProviders.Add(
            //new ExcludeBindingMetadataProvider(typeof(System.Version))));

            services.AddSwaggerGen(options =>
            {
                options.SwaggerDoc("v1", new Info { Title = "My Core API", Description =
"Swagger Core API", Version = "v1" });
            });
```

```
            services.AddMvc(options => options.MaxModelValidationErrors = 50);
        }

        // This method gets called by the runtime. Use this method to configure the HTTP
request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            app.UseSwagger();
            app.UseSwaggerUI(c =>
            {
                c.SwaggerEndpoint("v1/swagger.json", "Code API");
            });

            app.UseMvc();
        }
    }
}
```

Рішення з математичного забезпечення

Математична модель

Формально завданням може бути пошук рекомендованої вистави описаний наступним чином:

$$\forall u \in U,\ s'_u = arg\,max_{s \in S}\, h(u,s),$$

де U - набір відвідувачів, S - набір вистав, які можуть бути рекомендованими для відвідувачів, h - функція, яка визначає, наскільки деякі вистави відповідають деяким відвідувачам.

Тому необхідно вибрати таку виставу $s' \in S$, для якої значення задоволеності відівдувача $u \in U$ є максимальним.

Припустимо, що N - загальна кількість об'єктів, які можуть бути рекомендовані користувачеві, а також, що тег k_j характеризує виставу n_i, а $f_{i,\,j}$ - кількість входжень цього тегу до історії d_j. Тоді $TF_{i,\,j}$ (term frequency) - це відношення числа входжень тега до загальної кількості тегів історії відвіданих вистав, тобто:

$$TF_{i,j} = \frac{f_{i,j}}{max_z f_{z,j}}$$

Але якщо враховувати тільки частоту входження тега, то в більшості історій відвідуваня максимальна вага буде у найпоширеніших тегів, що, найімовірніше, призведе до неправильного оцінювання переваг користувача. Для уникнення подібного застосовується IDF_i (inverse document frequency) - величина, зворотна частоті входження тега до історії користувача. Визначаємо її як:

$$\mathrm{IDF}_i = log\frac{N}{n_i}$$

Таким чином, вага $w_{i,\,j}$ у тега k_i в об'єкті d_j позначається, як добуток частоти входження тега на зворотну частоту:

$$w_{i,j} = TF_{i,j} \times IDF_i$$

В такому випадку, контент об'єкта d_j визначаємо як:

$$Content\left(d_{j,i}\right) = (w_{1,j}, \ldots, w_{k,j})$$

Як зазначалося вище, рекомендаційні системи з фільтрацією вмісту пропонують вистави з урахуванням тих, які користувач відвідав раніше. Різні вистави порівнюються з тими, що були відвідані користувачем і з них рекомендуються ті, які мають максимальну схожість. Сукупність вистав, які користувач відвідав раніше, утворює визначену для користувача історію *ContentBasedProfile(u)* або, інакше кажучи, вектор ваг ($w_{u,\,1}$, ..., $w_{u,\,k}$), де кожна вага $w_{u,i}$ визначає важливість тега k_i для користувача u. Отже, *ContentBasedProfile(u)* і *Content(s)* можна уявити як TF-IDF вектори $\vec{w}_u$ і $\vec{w}_s$, при цьому функція задоволеності користувача *h(u,s)* може бути представлена як косинусний коефіцієнт векторів $\vec{w}_u$ і $\vec{w}_s$:

$$h(u,s) = \cos(\vec{w}_u, \vec{w}_s) = \frac{\vec{w}_u \vec{w}_s}{\|\vec{w}_u\| \|\vec{w}_s\|} = \frac{\sum_{i=1}^{K} \vec{w}_{i,u} \vec{w}_{i,s}}{\sqrt{\sum_{i=1}^{K} w^2{}_{i,u}} \sqrt{\sum_{i=1}^{K} w^2{}_{i,s}}}$$

, де K - загальна кількість тегів в системі.

Наведемо алгоритм підбору театральних заходів:

КРОК 1. З запиту бронювання місця, що приходить до контролера API з сайту від потенційного відвідувача отримати дані про виставу, що грається на театральній події.

КРОК 2. Отримати список відвідувачів, які були на заходах, де гралася ця ж вистава та історії їх відвідувань.

КРОК 3. Провести агрегацію історій відвідувачів з метою отримання агрегованого відвідувача з сумарною історією.

КРОК 4. Визначити вектор ваг тегів для агрегованого відвідувача.

КРОК 5. Розрахувати вектор задоволеності агрегованого користувача виставами.

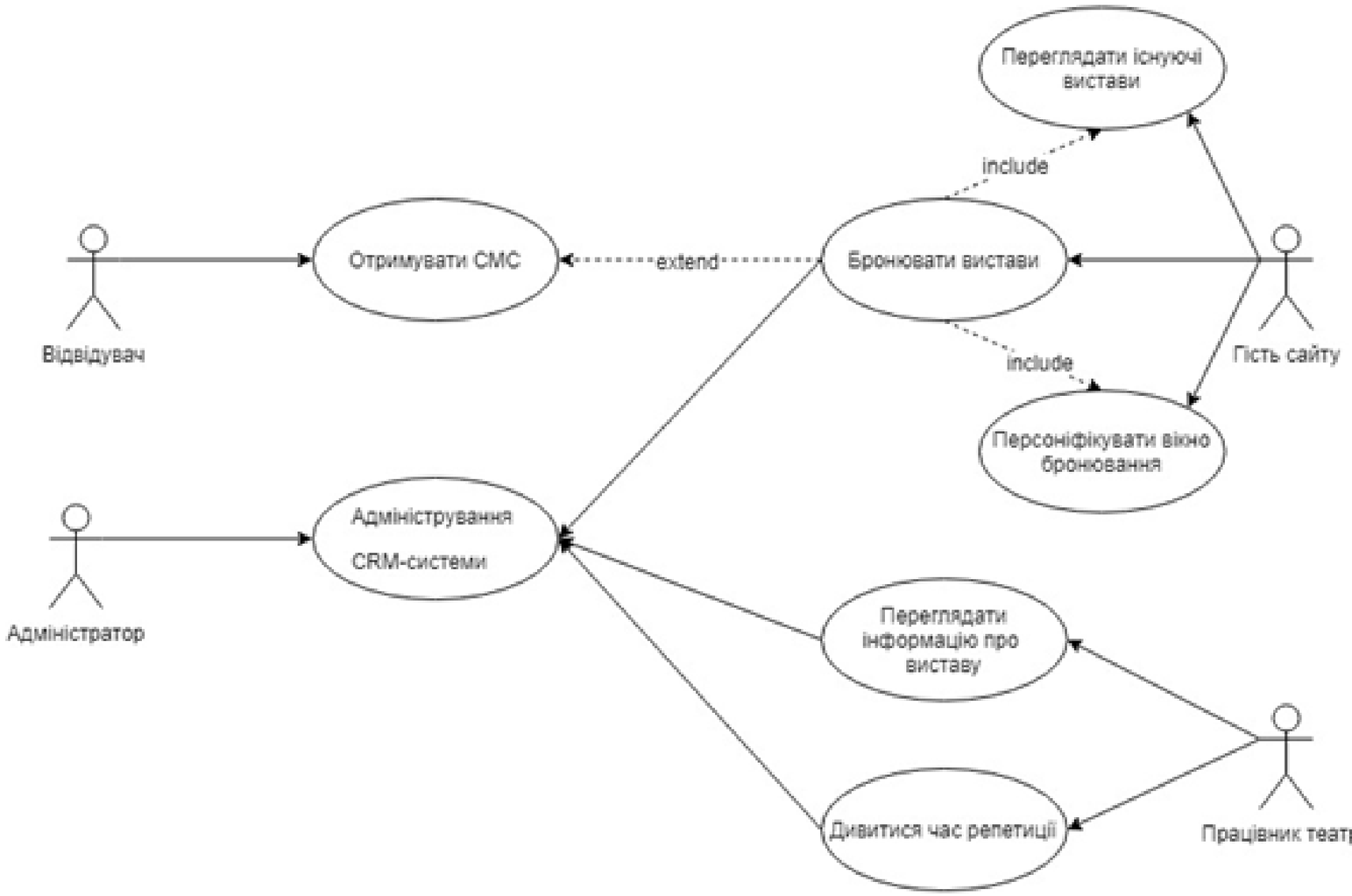
КРОК 6. Обрати 3 вистави з найбільшим значенням функціїї задоволеності.

Демонстраційний плакат до дипломного проекту

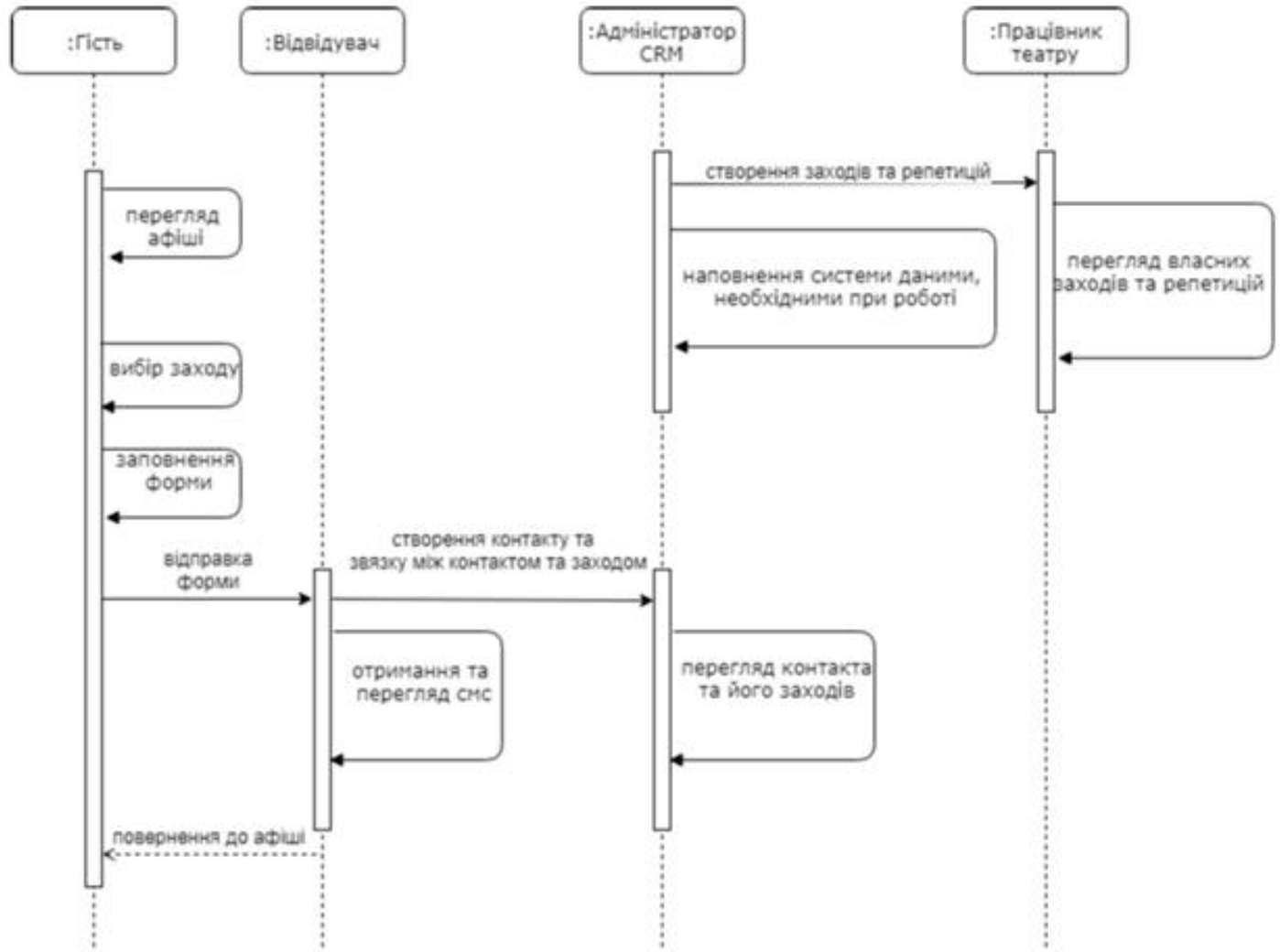
«Автоматизована рекомендаційна система підбору театральних подій»

Виконав студент гр. ІС-51 Штик В. Л.

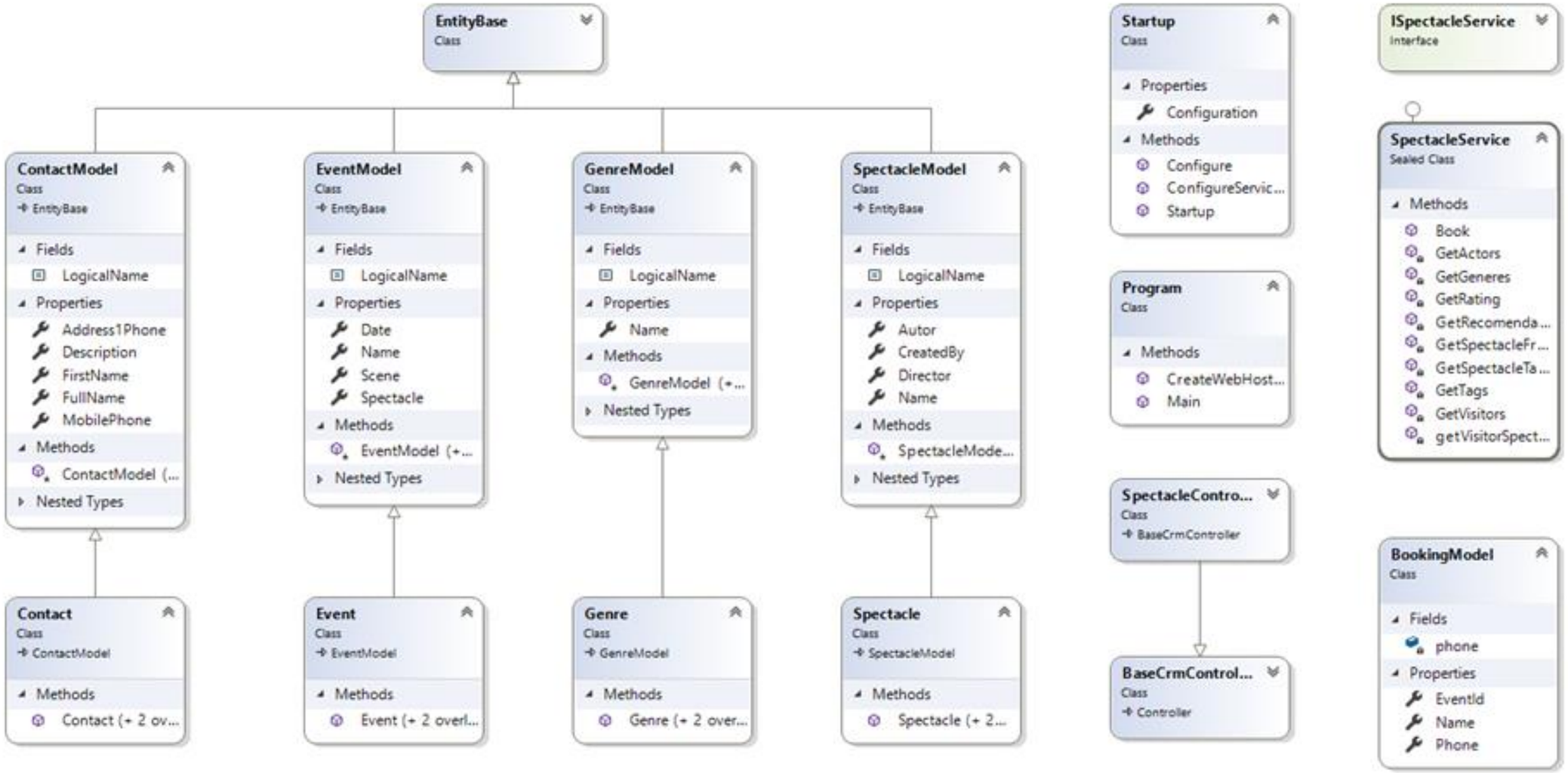
Керівник ДП Тєлишева Т.О.



					ДП ІС-5128.1181-с.ССВ		
						Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань			
Розробив		Штик В. Л.				Аркуш 1	Аркушів 1
Перевірив		Тєлишева Т.О.					
Т. кон.							
Н. кон.		Тєлишева Т.О.			Автоматизована рекомендаційна система підбору театральних подій	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51	
Затвердив		Тєлишева Т.О.					



					ДП ІС-5128.1181-с.ССД		
					Схема структурна діяльності	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Штик В. Л.				Аркуш 1	Аркушів 1
Перевірив		Тєлишева Т.О.					
Т. кон.							
					Автоматизована рекомендаційна система підбору театральних подій	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51	
Н. кон.		Тєлишева Т.О.					
Затвердив		Тєлишева Т.О.					



					ДП ІС-5128.1181-с.ССК		
					Схема структурна класів програмного забезпечення	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Штик В. Л.				Аркуш 1	Аркушів 1
Перевірив		Тєлишева Т.О.					
Т. кон.							
					Автоматизована рекомендаційна система підбору театральних подій	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51*	
Н. кон.		Тєлишева Т.О.					
Затвердив		Тєлишева Т.О.					

					ДП ІС-5128.1181-с.ССК		
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна компонентів програмного забезпечення*	Літера	Маса / Масштаб
Розробив		Штик В. Л.					
Перевірив		Тєлишева Т.О.				Аркуш 1	Аркушів 1
Т. кон.							
					Автоматизована рекомендаційна система підбору театральних подій	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51*	
Н. кон.		Тєлишева Т.О.					
Затвердив		Тєлишева Т.О.					